

PHP Web应用开发

娄不夜　张 军　编著

清华大学出版社

高等学校计算机课程规划教材

PHP Web 应用开发

娄不夜　张　军　编著

清 华 大 学 出 版 社
北　　京

内 容 简 介

本书以 Web 应用开发为背景，较为详细地介绍了 PHP 及其相关技术，内容包括 B/S 架构与 HTTP 协议、HTML 与 CSS、PHP 语言基础、PHP 函数和数组、字符串与正则表达式、MySQL 数据库基础、PHP 访问数据库、表单数据处理与验证、重定向与 PRG 模式、Cookie 与会话管理、文件上传与下载、PHP 面向对象程序设计、Ajax 与 jQuery 等。

本书立足基本理论和方法，注重实践与应用环节，对概念、原理和方法的描述力求准确、严谨，对例子和实例力求代码规范、面向实际应用。

本书可作为普通高等院校计算机、软件工程等相关专业的教材，也可作为 Web 应用开发者学习和使用 PHP 技术的参考书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

PHP Web 应用开发/娄不夜，张军编著. —北京：清华大学出版社，2017
（高等学校计算机课程规划教材）
ISBN 978-7-302-47584-2

Ⅰ. ①P…　Ⅱ. ①娄…　②张…　Ⅲ. ①网页制作工具– PHP 语言– 程序设计– 高等学校– 教材
Ⅳ. ①TP393.092　②TP312

中国版本图书馆 CIP 数据核字（2017）第 154969 号

责任编辑：汪汉友
封面设计：傅瑞学
责任校对：时翠兰
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn,010-62795954
印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**24　　**字　　数：**584 千字
版　　次：2017 年 10 月第 1 版　　**印　　次：**2017 年 10 月第 1 次印刷
印　　数：1～1500
定　　价：49.50 元

产品编号：072541-01

出 版 说 明

信息时代早已显现其诱人魅力，当前几乎每个人随身都携有多个媒体、信息和通信设备，享受其带来的快乐和便宜。

我国高等教育早已进入大众化教育时代，而且计算机技术发展很快，知识更新速度也在快速增长，社会对计算机专业学生的专业能力要求也在不断翻新，这就使得我国目前的计算机教育面临严峻挑战。我们必须更新教育观念——弱化知识培养目的，强化对学生兴趣的培养，加强培养学生理论学习、快速学习的能力，强调培养学生的实践能力、动手能力、研究能力和创新能力。

教育观念的更新，必然伴随教材的更新。一流的计算机人才需要一流的名师指导，而一流的名师需要精品教材的辅助，而精品教材也将有助于催生更多一流名师。名师们在长期的一线教学改革实践中，总结出了一整套面向学生的独特的教法、经验、教学内容等。本套丛书的目的就是推广他们的经验，并促使广大教育工作者更新教育观念。

在教育部相关教学指导委员会专家的帮助和指导下，在各大学计算机院系领导的协助下，清华大学出版社规划并出版了本系列教材，以满足计算机课程群建设和课程教学的需要，并将各重点大学的优势专业学科的教育优势充分发挥出来。

本系列教材行文注重趣味性，立足课程改革和教材创新，广纳全国高校计算机优秀一线专业名师参与，从中精选出佳作予以出版。

本系列教材具有以下特点。

1．有的放矢

针对计算机专业学生并站在计算机课程群建设、技术市场需求、创新人才培养的高度，规划相关课程群内各门课程的教学关系，以达到教学内容互相衔接、补充、相互贯穿和相互促进的目的。各门课程功能定位明确，并去掉课程中相互重复的部分，使学生既能够掌握这些课程的实质部分，又能节约一些课时，为开设社会需求的新技术课程准备条件。

2．内容趣味性强

按照教学需求组织教学材料，注重教学内容的趣味性，在培养学习观念、学习兴趣的同时，注重创新教育，加强“创新思维”和“创新能力”的培养、训练；强调实践，案例选题注重实际和兴趣度，大部分课程各模块的内容分为基本、加深和拓宽内容 3 个层次。

3．名师精品多

广罗名师参与，对于名师精品，予以重点扶持，教辅、教参、教案、PPT、实验大纲和实验指导等配套齐全，资源丰富。同一门课程，不同名师分出多个版本，方便选用。

4．一线教师亲力

专家咨询指导，一线教师亲力；内容组织以教学需求为线索；注重理论知识学习，注

重学习能力培养，强调案例分析，注重工程技术能力锻炼。

经济要发展，国力要增强，教育必须先行。教育要靠教师和教材，因此建立一支高水平的教材编写队伍是社会发展的关键，特希望有志于教材建设的教师能够加入到本团队。通过本系列教材的辐射，培养一批热心为读者奉献的编写教师团队。

清华大学出版社

前　言

PHP 是当今 Web 应用最流行的开发语言，它以简单性、开放性、低成本、安全性和适应性等深受 Web 程序员的青睐，被全世界越来越多的网站使用。全球 WWW 网站技术调查报告（W3Techs.com）的最新数据显示，世界上有 82.4%的网站使用 PHP 语言，在所有服务器端编程语言中占有绝对的优势。Facebook、Baidu、Wikipedia、Twitter 等著名的网站都在使用 PHP 技术。

Web（网）是英国人 Tim Berners-Lee 于 1990 年发明的。在 2000 年前后出现了“Web 应用”一词，它代表运行于 Web 服务器，可供用户通过 Web 在浏览器中进行访问的计算机应用软件。目前，Web 应用是计算机应用软件最主要的形式之一。

本书以 Web 应用开发为背景，较为详细地介绍了 PHP 及相关技术，内容包括 B/S 架构与 HTTP 协议、HTML 与 CSS、PHP 语言基础、PHP 函数和数组、字符串与正则表达式、MySQL 数据库基础、PHP 访问数据库、表单数据处理与验证、重定向与 PRG 模式、Cookie 与会话管理、文件上传与下载、PHP 面向对象程序设计、Ajax 与 jQuery 等。

全书立足基本理论和方法，注重实践与应用环节。对概念、原理和方法的描述力求准确、严谨，例子代码力求精简、规范。本书每章的最后都配有精选习题，便于读者复习、巩固、练习与提高。

本书还引入了一个较为完整的 Web 应用——教务选课系统。系统分管理员子系统、学生和教师子系统两部分。本书正文以数据处理和页面制作模块化为指导思想，介绍了管理员子系统的开发。子系统的开发介绍没有独立成章，而是随各章知识点的逐步介绍和推进，分步骤、分层次地展开，以实例的形式分布在有关章节中。学生和教师子系统被设计成 11 个实验题，以附录的形式放置在全书最后，供读者练习。

本书在编排时使用了一些符号和特殊处理，这里做简单说明：

（1）代码左边的行号是为了引用和讲述方便而增加的，不是代码的一部分。

（2）在语言成分的语法格式描述中：

- 符号“< >”表示该项由程序员按规则指定或定义。
- 符号“[ ]”表示该项为可选项。
- 符号“[ ]*”表示该项可重复 0 至多次。
- 符号“|”可以将两项或多项连接起来，表示选择其中一项。为标明第一项的开始处及最后一项的结尾处，可用符号“{}”将这些选项括起来。

需要注意的是，这些符号在有些语言成分中本身就有其特定的作用，例如“< >”在 HTML 标记语言中表示标签的开始和结束；“*”在 SQL SELECT 语句中表示所有列；“[ ]”在 PHP 中表示访问数组元素；“{}”在 PHP 中表示块语句的开始和结束，在 CSS 规则中表示声明块的开始和结束，等等。读者在阅读时还需要根据上下文来判断这些符号的具体含义。

本书提供相关的教学资源，包括教学课件以及所有例子和实例的源代码。欢迎读者从

清华大学出版社网站（http://www.tup.tsinghua.edu.cn）下载。

由于作者学识和水平有限，本书难免有错误和不妥之处，敬请广大读者批评指正。如果读者有好的建议或要求，请与作者联系，电子邮箱地址是 loubuye@163.com。

作 者

2017 年 9 月

目　　录

第 1 章　PHP 入门

本章主题：

- PHP 及其由来；
- 万维网基础（URL、HTTP、HTML）；
- PHP 标签与 PHP 代码块；
- 输出 HTML；
- 代码注释；
- PHP 工作原理；
- PHP 运行环境与开发工具。

PHP 是一种动态网页技术，可用于开发 Web 应用。本章首先介绍 PHP 及其由来、万维网的一些基础知识，然后介绍 PHP 标签、PHP 代码块、输出 HTML 等 PHP 的基本语法及使用。本章最后介绍 PHP 的运行环境、开发工具的安装以及 NetBeans IDE for PHP 的操作和使用。

1.1　PHP 及其由来

PHP 是一种应用广泛、开源、通用的脚本语言，主要作为动态页面和 Web 应用的服务器端脚本语言使用。 PHP 页面总体来说就是 HTML 页面，只是可以在其中嵌入一些 PHP 代码，以便动态产生一些内容。PHP 代码吸收了 C、Perl 等语言的语法，易于学习。PHP 的最大特点是非常容易入门、学习和使用，同时也为专业程序员提供了许多高级特性。

PHP 最初由丹麦人 Rasmus Lerdorf 于 1994 年发布。该版本已经包含了今天 PHP 的一些基本功能：拥有 Perl 样式的变量，能自动解析表单变量，可以嵌入 HTML。语法本身与 Perl 很相似，但还很有限、很简单，且稍显粗糙。

1997 年 PHP 2.0 正式发布。该版本具备了更多的编程语言特征，提供了内置的数据库访问、Cookies、用户自定义函数等功能。

1997 年，Andi Gutmans 和 Zeev Suraski 加入了 PHP 的开发，重新编写了语法分析程序，形成了 PHP 3.0 的基础。1998 年，PHP 3.0 正式发布。PHP 3.0 对 PHP 重新进行了命名，形成一个递归缩写，即 PHP 是“PHP: Hypertext Preprocessor”。

PHP 3.0 被认为是类似于当今 PHP 语法结构的第一个版本。1995 年，Rasmus Lerdorf 公开相关源代码并发布了第一个版本，一开始称为 PHP Tools（Personal Home Page Tools），后来又称为 PHP/FI（Personal Home Page/Forms Interpreter）。截至 1998 年末，有大约 10 万个网站使用了 PHP。在 PHP 3.0 的顶峰期间，Internet 上 10%的 Web 服务器上都安装了它。

PHP 3.0 发布不久，Andi Gutmans 和 Zeev Suraski 又开始对 PHP 解释器的核心代码进行重新编写，并把它命名为 Zend Engine。2000 年，使用了 Zend 引擎的 PHP 4.0 正式发布。

除了更高的性能以外，PHP 4.0 还引入了一些新的功能，比如支持更多的 Web 服务器、HTTP 会话、一些新的语言结构等。

PHP 5.0 于 2004 年 7 月发布，它的核心是 Zend 的 2 代引擎。PHP 5.0 除了在性能上有增强之外，还引入了许多新的特性，如改善了对面向对象编程的支持，PDO（PHP Data Objects）扩展等。

2015 年 4 月 16 日，PHP 开发团队宣布 PHP 5.6.8 可用。

2015 年 12 月 3 日，PHP 开发团队宣布 PHP 7 可用。PHP 7 使用了新版的 Zend 引擎，改善了性能并引入了许多新的特性。PHP 7 的速度是 PHP 5.6 的两倍。

1.2 Web 基础

Web 是 World Wide Web 的简称，又称 WWW 或 W3，中文译名为“万维网”，是目前因特网（Internet）上最主要的信息服务形式。万维网由许多互相链接的 HTML 文档等信息资源组成，这些信息资源又由遍布在互联网上的称为 Web 服务器的计算机管理，用户则可以通过客户端浏览器进行浏览。万维网的基本结构如图 1-1 所示。

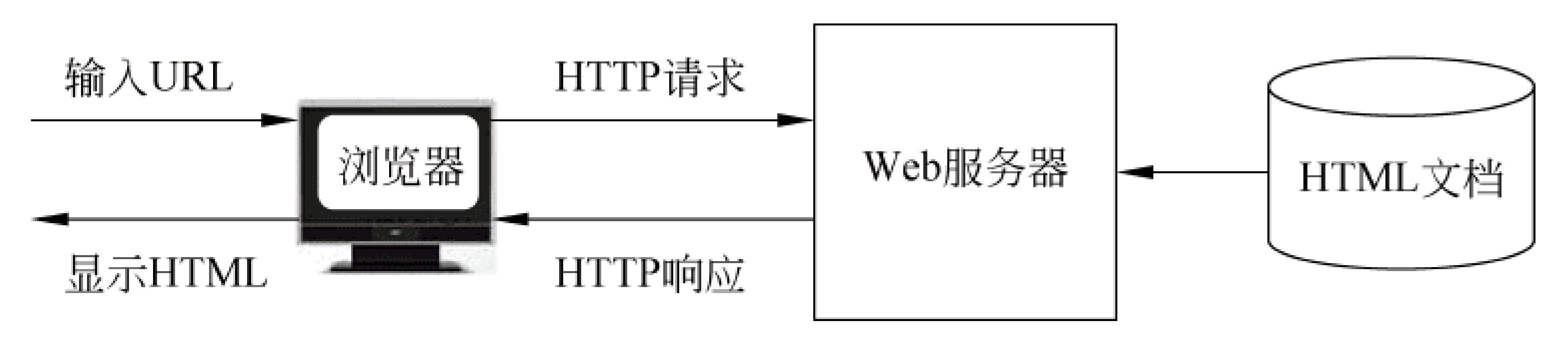


图 1-1　Web 基本结构

万维网技术最早由欧洲核子研究中心的蒂姆 • 伯纳斯-李（Tim Berners-Lee）于 1990 年提出，其最初的目的是为了解决各个研究项目组和科研人员之间的信息交流和信息共享问题，避免因信息交流不畅和丢失而使研究工作重复进行。万维网技术的核心包括统一资源定位符（URL）、超文本传输协议（HTTP）和超文本标记语言（HTML）。

1.2.1 URL

统一资源定位符（Uniform Resource Locator，URL），也称为网页地址或简称网址，是定位因特网上资源的标准地址。URL 的语法格式如下：

```
<协议类型>://<服务器名>[:<端口号>]/<路径>[?<查询串>]
```

其中参数含义如下。

（1）协议类型：指定传输信息所采用的网络协议。最基本的协议是 HTTP，有时也可能会用到其他协议，例如 HTTPS、FTP、MAILTO 等。

（2）服务器名：指定资源所在的 Web 服务器的域名，通常以.com、.net、.org、.gov、.cn 等后缀结尾。有时也可用 IP（Internet Protocal）地址来指定 Web 服务器。

（3）端口号：端口又指协议端口，是特定应用或进程作为通信端点的软件结构。端口号是一个无符号整数，范围为 0～65535。对于 Web 服务器提供的 HTTP 通信服务，默认端口号是 80，此时在 URL 中经常省略端口号。

（4）路径：用于指定资源，通常包括资源在 Web 服务器中的位置信息及名称。路径一般是区分字母大小写的。

（5）查询串：是一个经过编码的、由一组名称/值对（用&分隔）组成的字符串，表示在 HTTP 请求中发送的数据，也称为请求参数。例如：id=100&p=2。

1.2.2 HTTP

超文本传输协议（HyperText Transfer Protocal，HTTP）是一种建立在 TCP（Transmission Control Protocal）协议之上的属于应用层的网络协议，是万维网上数据通信的基础，适用于分布式、协作式的超媒体信息系统。

1. 请求-响应过程

HTTP 是一种无连接、无状态的协议。无连接并不是不需要连接，而是指每次连接仅限于一次请求-响应过程。下一次的请求-响应过程需要重新进行连接。无状态是指协议没有记忆约定，前后两次请求-响应过程是相互独立的，协议本身并不会依据上一次请求-响应的状态来处理下一次的请求-响应。

基于 HTTP 协议的请求-响应过程分 4 个步骤。

（1）建立连接。通过域名（或 IP 地址），客户端连接到服务器。

（2）发送请求。建立连接后，客户端把请求信息发送到服务器的端口上，完成请求动作。

（3）发送响应。服务器处理请求，然后向客户端发送响应信息。

（4）关闭连接。当响应发送完毕，服务器关闭连接。客户端也可以在完整接收响应之前，终止数据传输，关闭连接。

2. 请求信息

客户端向服务器发送的请求信息有较为固定的内容组成和格式，由请求行、请求头和请求体等组成。图 1-2 是请求信息的一个示例。

首先是请求行，它包含方法、请求 URI 和协议版本号 3 项内容，相互之间用空格分隔。

```
<方法> <请求 URI> <协议版本号>
```

其中，方法用于指定本次请求的性质，即本次请求要对指定的资源做何种操作，例如查询、添加、删除、更新等。目前，大多数浏览器仅支持 GET 和 POST 两种方法。一般来说，查询操作应该用 GET 方法，其他操作则可用 POST 方法。通常，采用 GET 方法的请求也称为 GET 请求，采用 POST 方法的请求也称为 POST 请求。

请求 URI（Uniform Resource Identifier）通常是 URL 中端口号后面的内容，即不包括其中的协议类型、服务器名和端口号，用于标识资源。这时，有关服务器的信息应该在请求头 Host 域中指定。

请求行的后面是请求头。请求头包含一些域，每个域占一行，由域名和域值组成，两者之间用冒号分隔。有些域可能有多个值。请求头用于指定本次请求及客户端浏览器的一些附加信息。

请求头的后面是一个空行（仅包含回车换行符），该空行表示请求头的结束。

请求信息的最后是可选的请求体。请求体的内容依据请求方法的不同而不同。对 POST

```
请求行：  POST /hello/hello.html HTTP/1.1
请求头：  Host: localhost:8080
          Accept-Language: zh-cn,zh
          Content-Type: application/x-www-form-urlencoded
          Content-Length: 30
          …
空行：
请求体：  username=zhang&password=123456
```

图 1-2　请求信息示例

请求，请求体一般仅包含一些请求参数。对 GET 请求，请求体往往是空的，此时请求参数包含在请求行的请求 URI 中。

3. 响应信息

与请求信息一样，服务器向客户端发送的响应信息也有固定的内容组成和格式。响应信息由状态行、响应头和响应体等组成，图 1-3 是响应信息的一个示例。

```
状态行：  HTTP/1.1 200 OK
响应头：  Server: GlassFish Server Open Source Edition 3.0.1
          Content-Type: text/html
          Content-Length: 232
          Date: Mon, 20 Feb 2012 09:55:09 GMT
          …
空行：
响应体：  <html> … </html>
```

图 1-3　响应信息示例

响应信息的第一行是状态行，它包含协议版本号、状态码及相应的状态描述信息，相互之间用空格分隔。

<协议版本号> <状态码> <状态描述>

状态行中的状态码由 3 位数字组成，其中第一位数字是对响应状态的一个分类。下面是 5 类状态码的一个总体描述。

（1）1××：消息，请求已被接收，继续处理。

（2）2××：成功，请求已成功被服务器接收、理解、接受并处理。

（3）3××：重定向等，客户端需要后续操作才能完成这一请求。

（4）4××：客户错误，请求含有词法错误或者无法被执行。

（5）5××：服务器错误，服务器在处理某个请求时发生错误。

状态行后面是响应头。与请求头类似，响应头也由一些域组成，用于指定本次响应以及服务器的一些附加信息。

响应头的后面是一个空行（仅包含回车换行符），该空行表示响应头的结束。

响应信息的最后是响应体，是响应内容本身，如客户请求的 HTML 文档内容。

1.2.3　HTML

在构建 Web 页时，超文本标记语言（HyperText Markup Language，HTML）是一种主要的标记语言。

HTML 文档是一种文本文件，由要显示的内容数据和 HTML 标签组成。HTML 标签用于指定文档的结构、内容的显式格式。每个 HTML 标签由一个小于号、标签名称、属性和一个大于号构成。一般情况下，标签是成对出现的，即以起始标签开始、以结束标签结尾，两者之间的内容是标签的作用范围。

HTML 允许在 Web 页中嵌入图像、对象，以及用于接收请求数据的表单。HTML 允许在页面中嵌入客户端脚本代码（例如 JavaScript），能够影响页面的行为。HTML 支持 CSS 技术，以便定义页面内容的外观和布局。

Web 浏览器的作用是接收 HTML 文档、解析 HTML 标签，产生符合阅读和浏览习惯的 Web 页面。

1.3　在 Web 页中嵌入 PHP 代码

PHP 代码可以被嵌入 HTML 页面中。这些含有 PHP 代码的 Web 页面被称为 PHP 页面。PHP 页面文件的扩展名是.php。当客户端用户请求 PHP 页面时，Web 服务器不是把 PHP 页面直接送往客户端，而是先将 PHP 页面文档委托给 PHP 解释器处理，然后再将处理结果送往客户端。

1.3.1　PHP 标签

PHP 代码以代码块的形式嵌入到 HTML 页面中。一个 PHP 代码块以“<?php”开始，以“?>”结束，其中“<?php”和“?>”称为 PHP 标签，两个标签的各字符之间都不能插入空格。

一个 PHP 代码块可以包含多条 PHP 语句，每条语句以分号（;）结尾。处于结束标签（?>）之前的最后一条 PHP 语句可以省略分号。

【例 1-1】 PHP 标签。代码如下：

```
<html>
   <head>
      <title>PHP 标签 1</title>
   </head>
   <body>
      <?php
         echo '<p>Hello World</p>';
      ?>
   </body>
</html>
```

这个例子包含一个 PHP 代码块，其中仅包含一条 PHP 语句，即 echo 语句。该语句可

以输出指定的字符串。

PHP 解释器在处理一个 PHP 页面文档时，对 PHP 标签外的内容（HTML 标签）只是简单原样输出，对 PHP 标签内的 PHP 语句则进行解释和执行。其效果是，PHP 标签外的内容和 PHP 代码执行产生的内容按顺序组成一个完整的 HTML 页面。如果一切正常，上面的 PHP 页面经过 PHP 解释器处理后将产生如下的 HTML 页面代码：

```
<html>
  <head>
    <title>PHP 标签 1</title>
  </head>
  <body>
    <p>Hello World</p>
  </body>
</html>
```

最终，这个由 PHP 解释器处理产生的 HTML 页面代码将由 Web 服务器送往客户端，经浏览器解析后进行呈现。

1.3.2 其他风格的 PHP 标签

PHP 标签有 4 种不同的风格。

1. XML 风格

上述 PHP 标签（<?php 和?>）是 XML 风格的，它是 PHP 推荐使用的默认标签风格。除此之外，还有另外 3 种风格可供选择。

2. SCRIPT 风格

这种风格类似 JavaScript 的语法格式，使用<script>标签。例如：

```
<script language="php"> echo '<p>Hello World</p>'; </script>
```

3. 简短风格

这种风格使用更简短的语法，以“<?”开始，以“?>”结束，省略了默认风格中的 php 字样。例如：

```
<? echo '<p>Hello World</p>'; ?>
```

4. ASP 风格

这种风格与 ASP（Active Server Pages）的标签风格相同，以“<%”开始，以“%>”结束。例如：

```
<% echo '<p>Hello World</p>'; %>
```

PHP 标签的 4 种风格具有相同的作用，都用于界定一个 PHP 代码块。其中，默认风格和 SCRIPT 风格总是可用的。也就是说，所有 Web 服务器上的 PHP 解释器在任何时候都支持这两种风格。对简短风格和 ASP 风格的可用性，服务器管理员通常可以通过某种途径进行设置。例如，通过设置 PHP 配置文件 php.ini 中的 short_open_tag 或 asp_tags 项可以改变简短风格或 ASP 风格的可用性。

1.3.3　嵌入多个代码块

一个 PHP 页面可以包含多个 PHP 代码块，PHP 代码与 HTML 代码可以交替出现。一个 PHP 代码块可以嵌入在两个 HTML 标签之间，也可以出现在一个 HTML 标签内。

【例 1-2】 一个 PHP 页面包含了多个 PHP 代码块。代码如下：

```
<html>
   <head>
      <title>PHP 标签 2</title>
   </head>
   <body>
      <?php
         $date = date("Y 年 m 月 d 日");
      ?>
      <p>今天是：<?php echo $date; ?>!</p>
   </body>
</html>
```

在第 1 个 PHP 代码块中，通过 date 函数返回一个表示当前日期的字符串，并赋给了变量$date。第 2 个代码块通过 echo 语句输出当前日期。可见，前面代码块中声明的变量能够被记住，可以在后面的代码块中使用，例如本例中的$date 变量。这里，第 2 个代码块出现在 HTML 的 p 标签内。

一个代码块可以包含多条 PHP 语句。反之，有些 PHP 语句（流程控制语句）也可以分散在多个代码块内完成。

【例 1-3】 一个 PHP 语句横跨多个 PHP 代码块。代码如下：

```
<?php
   $n = 15;
   if($n%2 == 0) {
?>
   这是一个偶数。
<?php } else { ?>
   这是一个奇数。
<?php }; ?>
```

这里共有 3 个代码块。if 语句从第 1 个代码块开始，一直到第 3 个代码块结束。两个代码块之间的内容是静态的、可直接输出的非 PHP 代码。当然，也可以在 PHP 代码块内用 echo 等语句输出这些静态的内容。但是，如果这些静态的内容是大段的，那么将它们处理成非 PHP 代码通常比用 echo 等语句输出会更有效率。

另外，并非所有 PHP 标签外的非 PHP 代码都一定会被输出。在该例中，由于 if 条件不成立，所以第 1 个静态文本块（即文本“这是一个偶数。”）不会被输出。

如果一个 PHP 页面的最后内容是一个 PHP 代码块，或者这个页面由纯的 PHP 代码组成（只有一个 PHP 代码块，没有非 PHP 代码），那么页面中最后一个 PHP 结束标签（?>）

是可以省略的。此时，最后一条 PHP 语句末尾的分号就不能省略。这样做的好处是，PHP 解释器将忽略文件末尾的那些空格、回车换行符等空白符号，而不是将这些空白符号作为非 PHP 代码输出。

最后需要注意，该例的第 1 个代码块中最后一个花括号是不可少的，否则 PHP 解释程序会认为该 if 语句就此结束。第 2 个代码块中最后一个花括号也同样是不可少的，否则 PHP 解释程序会认为该 else 子句就此结束。

1.4 输出 HTML

PHP 代码块中的代码在 Web 服务器端解释和执行，在执行过程中可以动态产生并输出一些数据或 HTML 标签。这些输出内容构成了 HTTP 响应体的一部分。在 PHP 中，一般用 print 或 echo 语言结构来输出这些数据或 HTML 标签。

print 语言结构的语法格式如下：

```
int print <$arg>
```

或

```
int print(<$arg>)
```

print 用于输出一个字符串。如果参数不是字符串，而是其他类型的数据，系统会自动将其转换为字符串然后输出。print 总是返回整数 1。

echo 语言结构的语法格式如下：

```
void echo <$arg> [,<$arg>]*
```

或

```
void echo(<$arg>)
```

echo 可以输出 1 个或多个字符串，但采用函数形式时只能输出一个字符串。若输出多个字符串，各字符串按顺序一个挨着一个连续输出。与 print 一样，如果参数不是字符串，系统会自动将其转换为字符串然后输出。

1.5 代码注释

PHP 注释出现在 PHP 代码块内，用于对 PHP 代码进行说明，以提高代码的可读性。无论是代码的编写者还是软件的维护者，代码注释都是非常重要的。需要注意，注释不允许嵌套，即对注释本身无须再注释。

PHP 注释包括单行注释和多行注释两类。

1.5.1 PHP 单行注释

单行注释从“//”开始，止于行尾或当前 PHP 代码块的尾部。通常用于对当前行代码做

简单说明，例如：

```
<?php
   echo "<h2>example1</h2>";     // output: <h2>example1</h2>
?>
<p><?php echo 'example2';        // output: example2 ?></p>
```

代码中包含两个单行注释，第 1 个注释止于行尾，第 2 个注释止于所在的代码块的尾部。

1.5.2 shell 风格单行注释

该种单行注释与 UNIX 中的 shell 语法一致，即用“#”表示注释的开始。除此之外，它与用“//”进行单行注释没有其他区别。上面例子也可以采用 shell 风格单行注释：

```
<?php
   echo "<h2>example1</h2>";     # output: <h2>example1</h2>
?>
<p><?php echo 'example2';        # output: example2 ?></p>
```

1.5.3 PHP 多行注释

多行注释以“/*”开始，并以“*/”结束。通常用于较长的说明，放置在被说明代码的前面。例如：

```
<?php
/*
* This is a detailed explanation of something that
* should require several paragraphs of information.
*/
…（被注释代码）
?>
```

这里，内部注释行前面的星号可以使注释范围更加醒目，当然去掉也无妨。

1.5.4 PHP 文档注释

PHP 文档注释以/**开始并以*/结束，一般用来注释类、函数（或方法）等程序模块，并放置在这些被说明模块的前面。文档注释往往会包含一些固定格式的元素，这些元素以@开头。如@author 标明开发该模块的作者，@version 标明该模块的版本和日期，@param 标明函数（或方法）的参数，@return 说明函数（或方法）的返回值等。

下面是 PHP 文档注释的一个例子：

```
/**
 * 检测$year 表示的年份是否为闰年。
 * @param int $year 定义被检测的年份。
 * @return boolean 若$year 为闰年，返回 true，否则返回 false。
```

```
 */
function isLeap($year) {
   return $year%400===0 || $year%4===0 && $year%100!==0;
}
```

1.5.5　HTML 注释

PHP 注释用来说明 PHP 代码，并不会送往客户端。HTML 注释用于说明 HTML 代码，会被送往客户端浏览器。如果用户在客户端浏览器查看 HTML 源文件，可以看到这些 HTML 注释。

HTML 注释以<!--开始、并以-->结束，可以持续多行。例如：

```
<!--这里是注释 -->
```

这里，开始标签（<!--）和结束标签（-->）的各字符之间一般不应该有空格。

1.6　PHP 工作原理

一个基于 PHP 技术创建的 Web 应用可被称为 PHP 应用。PHP 应用一般由 HTML 页面、CSS 文件、JavaScript 文件等静态资源和 PHP 页面（PHP 文件）等动态资源组成。

与 Web 静态资源由客户端解析、呈现或运行不同，PHP 页面（PHP 文件）在服务器端被解析、处理和运行，并动态产生响应。

PHP 页面与静态资源的上述差异，也反映在其请求与响应过程中。图 1-4 是引入 PHP 页面和 PHP 解释器后的 Web 结构示意图。

其请求-响应的基本过程如下：首先客户通过浏览器向 Web 服务器发送一个 HTTP 请求。接下来 Web 服务器接受客户请求并判断请求的是 Web 静态资源还是 PHP 动态资源。如果客户请求的是一个 Web 静态资源，Web 服务器可以直接从硬盘读取该资源并产生 HTTP 响应发往客户端。如果客户请求的是一个 PHP 动态资源，Web 服务器将读取该资源并将其委托给 PHP 解释器处理。PHP 解释器通过预编译、解释、运行等过程处理 PHP 页面或 PHP 文件，并产生 HTML 响应内容。最后由 Web 服务器向客户端发送 HTTP 响应。

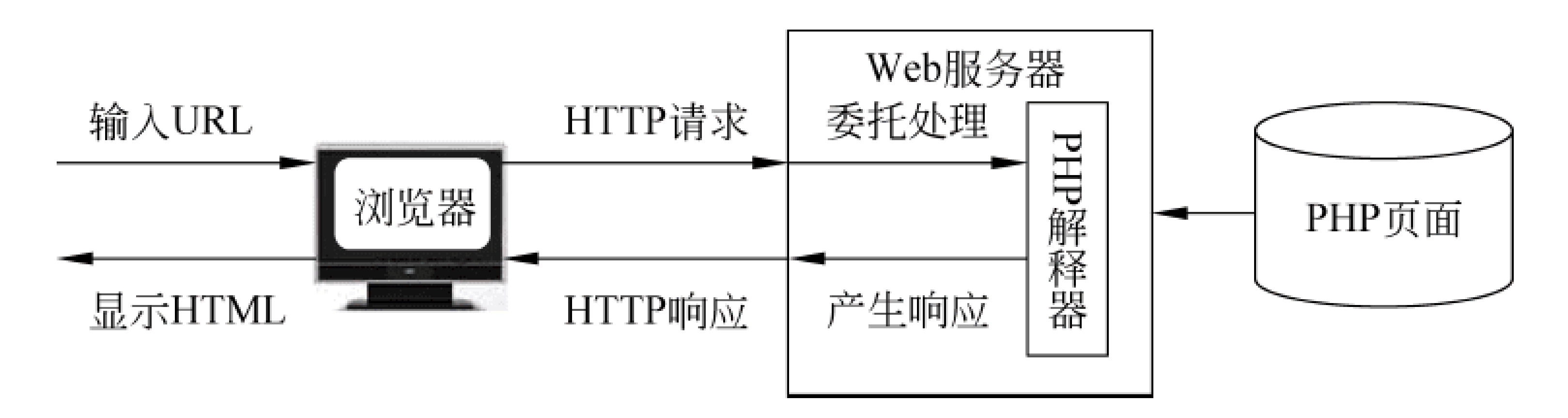


图 1-4　PHP 页面的请求与响应

1.7　运行环境与开发工具

PHP 页面或 PHP 文件需要在 PHP 解释器的支持下才能预编译、解释和运行，而 PHP 解释器又需要与 Web 服务器配合才能完成对客户端请求的处理与响应。另外，PHP 应用通

常需要用到数据库系统实现对应用数据的保存、更新和查询，所以 PHP 解释器、Web 服务器和数据库管理系统被统称为 PHP 的运行环境。

除了 PHP 运行环境，还需要有相应的开发工具。它可以帮助开发者更方便、更快捷地开发所需的 PHP 应用。常见的 PHP 开发工具包括 Zend Studio、EclipsePHP Studio（EPP）、NetBeans IDE for PHP 等。

1.7.1 PHP 运行环境

本书采用由第三方提供的 XAMPP（Apache+MySQL+PHP+PERL）软件包，可以方便、快捷地同时安装构成 PHP 运行环境的 PHP、Apache 和 MySQL 等软件，支持在 Windows、Linux、Solaris、Mac OS 等多种操作系统下安装使用。

Apache 是 Apache HTTP Server 的简称，是 Apache 软件基金会的一个开放源代码的 Web 服务器软件，可以运行在几乎所有广泛使用的计算机平台上。由于其速度快、性能稳定、跨平台和安全性等特点，Apache 被广泛使用，是目前最流行的 Web 服务器软件之一。

MySQL 是一款开放源代码的中小型关系数据库管理系统，被广泛应用于 Web 应用的开发中。它与 PHP 和 Apache 组合，可构成良好的 Web 应用运行环境。MySQL 支持在 Windows、Linux、Solaris、Mac OS 等多种操作系统下安装使用，为包括 C、C++、Python、Java、Perl、PHP 在内的多种编程语言提供 API。本书第 7 章和第 8 章会介绍 MySQL 的具体使用。

XAMPP 软件包可以从以下网站免费下载：

- http://sourceforge.net/projects/xampp/；
- https://www.apachefriends.org/zh_cn/download.html。

本书选用的 XAMPP 软件包是 Windows 平台的 XAMPP 5.6.8 版本，下载的安装程序文件是 xampp-win32-5.6.8-0-VC11-installer.exe。

双击上面的安装程序文件就可以进入带向导的安装过程了。首先会显示安装程序的欢迎页面，接下来只需根据安装向导的提示进行操作即可，期间可以指定一个安装位置。通常，所有的软件都安装在名为 xampp 的文件夹下，其中包含 htdocs、mysql 等子文件夹。

安装完成后，Windows 操作系统的开始菜单会有一个 XAMPP Control Panel 菜单项，单击后可以打开 XAMPP 控制面板，如图 1-5 所示。用户可以单击 Start 或 Stop 按钮来启动或关闭机器上 Apache 和 MySQL 服务器进程。其中，Apache 的端口号默认为 80，SSL 的端口号默认为 443，MySQL 的端口号默认为 3306。如果出现端口号冲突启动不了，可以重新设置相关的端口号。单击 Apache 右侧的 Config 按钮，打开 httpd.conf 文件，可以设置 Apache 端口号；打开 httpd-ssl.conf 文件，可以设置 SSL 端口号。单击 MySQL 右侧的 Config 按钮，打开 my.ini 文件，可以设置 MySQL 端口号。

单击 Quit 按钮可以退出 XAMPP 控制面板，但在退出控制面板之前，应先关闭运行着的 Apache 和 MySQL 服务器。

1.7.2 PHP 开发工具

本书采用 NetBeans IDE for PHP 作为 PHP 应用的开发工具。NetBeans IDE 是一种免费、开源的集成开发环境。它支持开发者利用 Java、C/C++、PHP、JavaScript、Groovy 等语言和技术开发专业的桌面应用、企业级应用、Web 应用、移动应用等。与 NetBeans IDE 相比，

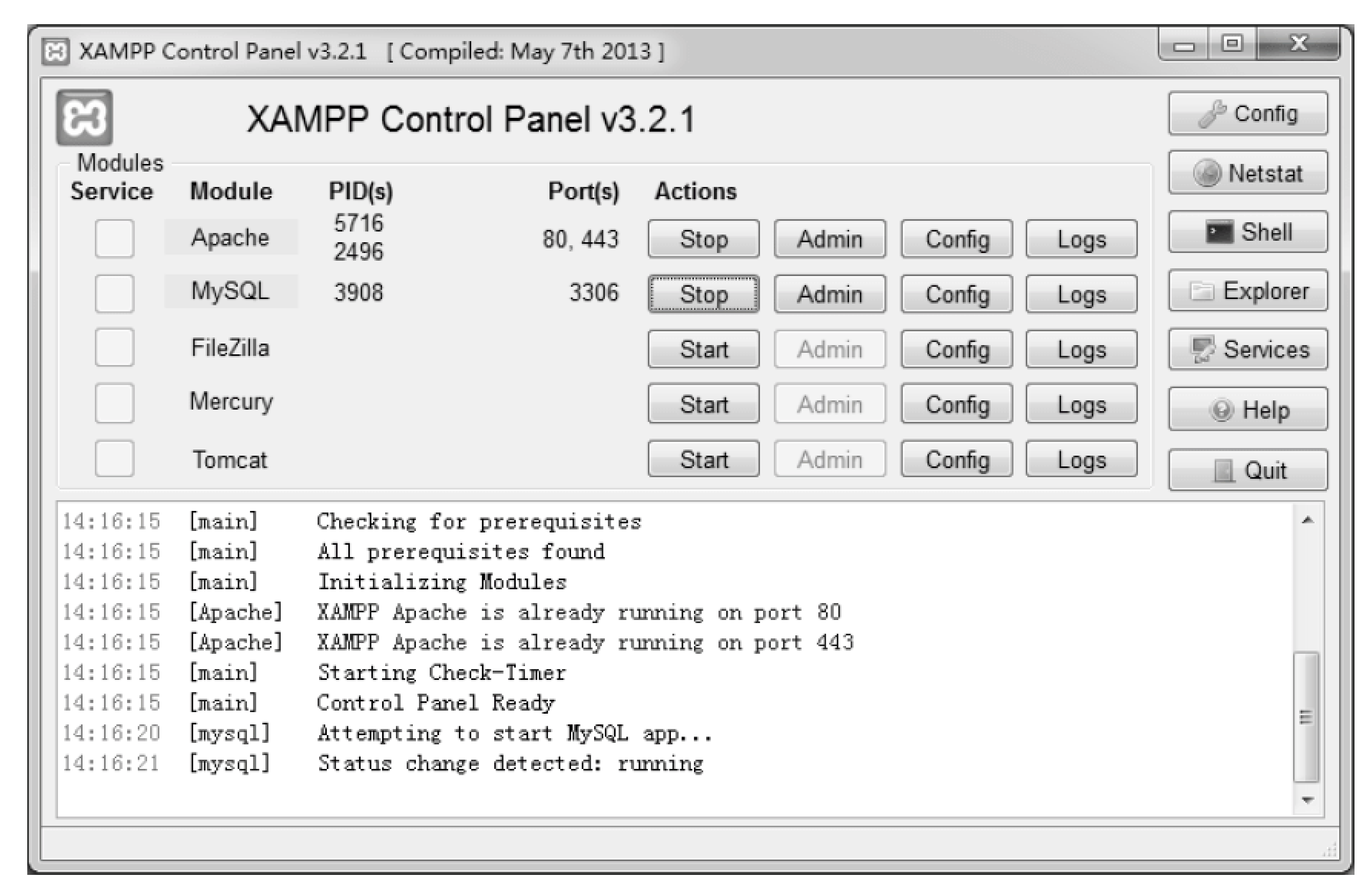


图 1-5　XAMPP 控制面板

NetBeans IDE for PHP 保留了 NetBeans IDE 的核心模块，但去除与 PHP 开发无关的模块。

NetBeans IDE for PHP 可以从 NetBeans 的官方网站（http://netbeans.org/）上下载。本书使用的 NetBeans IDE for PHP 是简体中文、Windows 平台的 NetBeans IDE 8.0.2 版本，下载的安装程序文件是 netbeans-8.0.2-php-windows.exe。

安装和运行 NetBeans IDE 之前，需要先安装 JDK。通常，不同版本的 NetBeans IDE 需要相应版本的 JDK。例如，NetBeans IDE 8.0.2 需要 JDK 6 或之后的版本。

安装 JDK 后，就可以安装 NetBeans IDE for PHP 了。要安装 NetBeans IDE for PHP，只需双击运行安装程序文件，然后根据安装向导的提示进行操作即可。在安装过程中，向导会引导选择合适的 JDK，用户可以设定该开发工具的安装位置。安装后，在桌面和开始菜单中通常会创建相应的快捷方式和菜单项。

安装好 NetBeans IDE for PHP 后，就可以开始工作了。在开发第一个 PHP 应用之前，需要先熟悉一下 NetBeans IDE for PHP 界面的基本组成。初始启动 NetBeans IDE 时，其界面主要包含一个显示于编辑器窗格内的“起始页”。大多数情况下，NetBeans IDE for PHP 界面大致如图 1-6 所示。

在 NetBeans IDE 界面中，除菜单栏和工具栏外，主要由一些位置和大小可变的窗格组成，其中编辑器窗格通常位于界面的中间，用于显示和编辑文档内容。每个在编辑器窗格打开的文档都有一个标签，通过这些标签可以在不同文档间快速切换。其他窗格可以包含一个或多个不同用途的窗口，每个窗口在窗格中也都有一个标签。下面是一些常用的窗口。

1.“项目”窗口

“项目”窗口是项目源的主入口点，是显示项目重要内容的逻辑视图。项目可以是 Java 应用程序、EJB 模块、Web 应用程序等。在这里，主要是 PHP 应用。

2.“文件”窗口

“文件”窗口显示基于目录的项目视图，其中包括“项目”窗口中未显示的任何文件和文件夹。

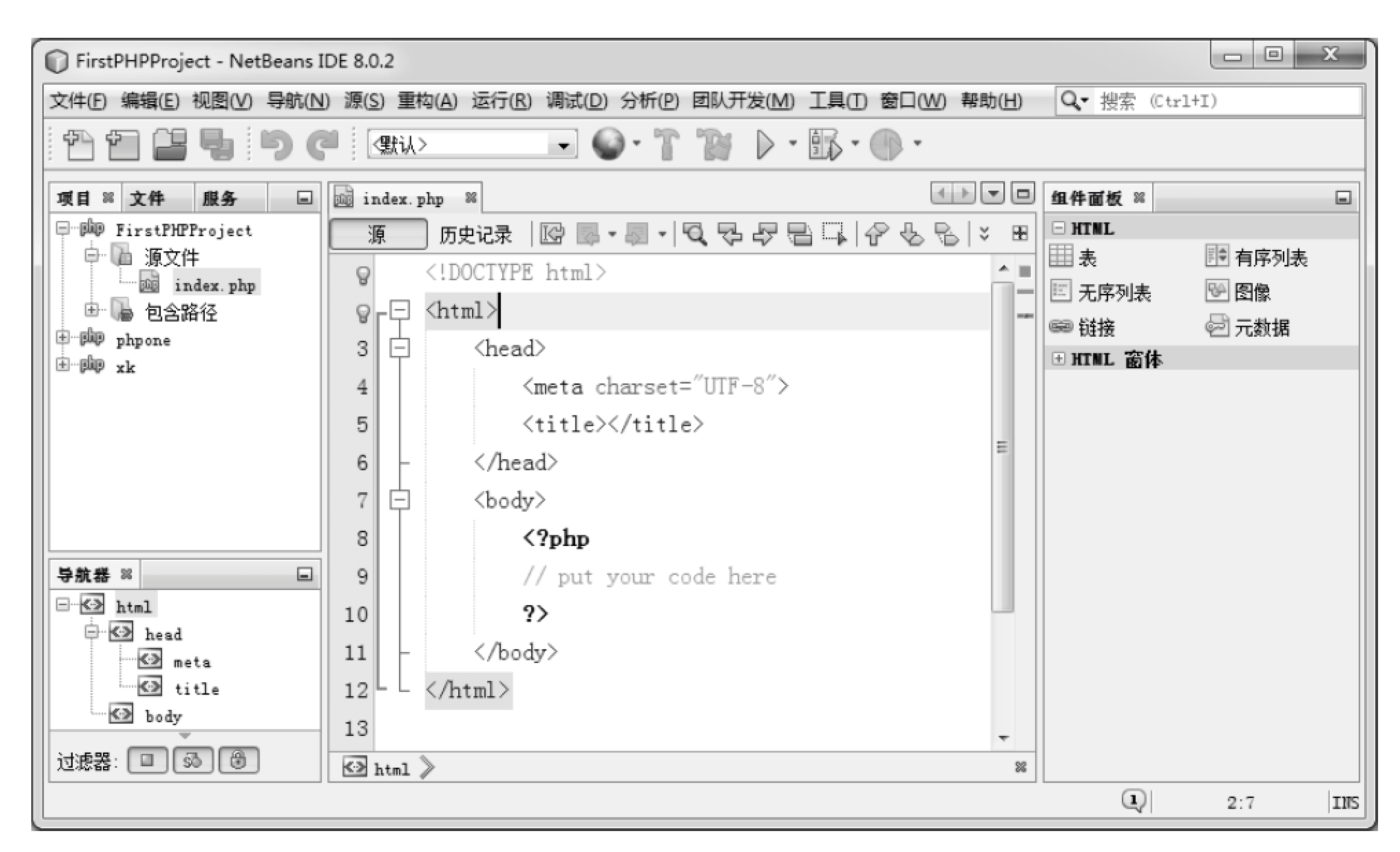


图 1-6　NetBeans IDE 界面

3.“导航器”窗口

“导航器”窗口提供了当前选定文件的简洁视图，并且可以简化文件不同部分之间的导航。在“导航器”窗口中双击某个元素，编辑器窗格中的光标就会移至该元素。

4.“组件面板”窗口

“组件面板”窗口主要包含 HTML 及 HTML 表单的一些组件标签。可以帮助用户在正在编辑的 HTML 文件、PHP 文件中快速添加这些标签。

对各种窗口，用户可以根据需要打开或关闭。要打开一个窗口，可以从“窗口”菜单中选择相应的菜单项或子菜单项。要关闭一个窗口，只需单击该窗口标签右端的“关闭窗口”按钮。某些窗口会在执行相关任务时自动出现。

通过鼠标拖放窗口标签，可以移动窗口。窗口可以在同一窗格内移动，也可以在不同窗格间移动。但不能将窗口移动到编辑器窗格内。编辑器窗格只能包含文档，不能包含其他窗口。反过来，也不能把文档标签移动到编辑器窗格之外。

1.8　使用 NetBeans IDE for PHP

这里介绍在 NetBeans IDE for PHP 中开发 PHP 应用的一些基本操作。

1. 创建 PHP 应用

创建 PHP 应用的具体步骤如下（在“项目”窗口中进行）。

（1）选择“文件”|“新建项目”菜单项，打开“新建项目”对话框。

（2）选择项目类型：“PHP”|“PHP 应用程序”。

（3）指定项目名称和位置。比如把项目名称指定为 FirstPHPProject，把存放项目文件的源文件夹指定为 C:\xampp\htdocs\FirstPHPProject，如图 1-7 所示。这里假设 XAMPP 软件包安装在 C:\xampp 下。



图 1-7　设置项目名称和位置

这里，FirstPHPProject 是存放项目文件的源文件夹。这个文件夹应该位于 Apache 服务器的文档根目录下。文档根目录可以在服务器配置文件中设置。默认情况下，服务器的文档根目录是 XAMPP 安装目录下面的 htdocs 子目录。

（4）设置运行配置。单击“下一步”按钮，进入“运行配置”，这里需要选择项目运行方式，指定运行项目的 URL，如图 1-8 所示。

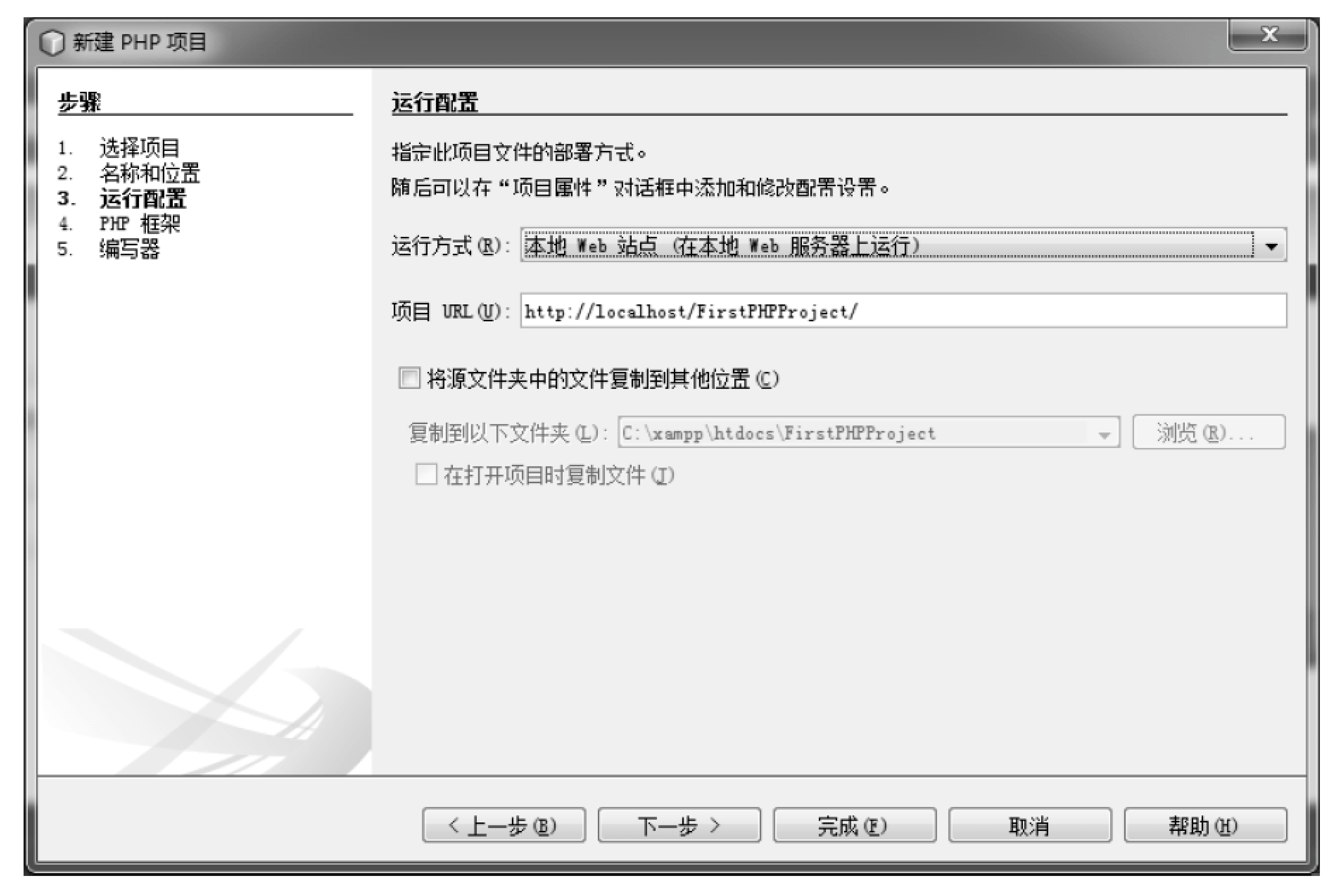


图 1-8　设置运行配置

（5）单击“完成”按钮，完成应用项目的新建。

新创建的 PHP 应用项目会自动包含一个名为 index.php 的 PHP Web 页，位于项目内的源文件结点下。

2. 新建文件

要为 PHP 应用添加文件，可按以下步骤操作（在“项目”窗口中进行）。

（1）单击选择项目结点。

（2）选择“文件”|“新建文件”菜单命令，打开“新建文件”对话框。

（3）选择文件类型，例如“PHP ”|“PHP 文件”、“PHP”|“PHP Web 页”、“HTML5”|“HTML 文件”等。

（4）指定文件名，一般不需要指定扩展名。

（5）单击“完成”按钮。

新创建的文件一般位于项目内的“源文件”结点下。

3. 打开和关闭 PHP 应用

可以选择“文件”|“打开项目”菜单项打开 PHP 应用，然后在“打开项目”对话框中定位并选择 PHP 项目文件夹，最后单击“打开项目”命令按钮打开该 PHP 应用。

PHP 项目文件夹应该位于 Apache 服务器的文档根目录下。默认情况下，服务器的文档根目录是 XAMPP 安装目录下面的 htdocs 子目录。

要关闭 PHP 应用，可以在“项目”窗口中右击项目结点，在弹出的快捷菜单中选择“关闭”项。关闭项目后，项目文件并没有从硬盘上删除，需要时可以再次打开它。

4. 访问页面

在访问页面（HTML 页、PHP Web 页、PHP 文件）之前，首先要确保 Apache 服务器已经启动。如果页面涉及数据库操作，还要事先启动 MySQL 服务器。

要访问一个页面，可以选择下面一种操作方法：

（1）在“项目”窗口中，右击要运行的页面文件结点，在弹出的快捷菜单中选择“运行”项。

（2）如果要运行的页面文件正好显示在编辑器窗格内，那么可以右击编辑区，然后在弹出的快捷菜单中选择“运行文件”项。

当访问 PHP 应用项目的页面时，NetBeans IDE for PHP 将完成以下工作。

（1）对项目中文件的任何修改，自动进行保存。

（2）通过浏览器向 Apache 服务器发出请求，请求指定的页面。如果浏览器还没有打开，那么会先自动打开浏览器。

习 题 1

1. 写出下面 PHP 页面的呈现结果

（1）

```
<html>
  <body>
    <?php
      $greeting = "Hello";
```

```
        $name = "LiMing"
    ?>
    <p>段落 1</p>
    <?php
        echo $greeting, $name;
    ?>
    <p>段落 2</p>
  </body>
</html>
```

（2）

```
<html>
  <body>
    <?php
        $color = "red";
    ?>
    <p style="color:<?php echo($color) ?>">这是一个段落</p>
  </body>
</html>
```

（3）

```
<html>
  <body>
    <?php
        $n = 1;
        if($n == 0)
    ?>
    <p>yes</p>
    <p>end</p>
  </body>
</html>
```

2. 简答题

（1）简述万维网的基本结构及工作原理。

（2）简述 PHP 的工作原理。

（3）举例说明在 PHP 文件中可以使用的几种注释。

第 2 章　HTML 与 CSS 简介

本章主题：

- HTML 标签与元素；
- HTML 文档基本结构；
- HTML 基本元素；
- HTML 列表、表格和表单；
- CSS 基本语法；
- CSS 选择器；
- 定义和使用 CSS；
- 常用 CSS 属性和属性值。

HTML 和 CSS 是制作 Web 网页的语言和工具，也是 Web 应用的基础。在 Web 应用中，网页是用户界面，既用来呈现用户所需的信息，又需要接收用户输入的数据，以便在服务器端作进一步的处理。

HTML 主要用于表示文档的结构，例如标题、段落、列表、表格等，其中每一种元素通常也会有默认的呈现格式。CSS 主要用于设置文档内容的呈现格式以及页面布局等。本章首先介绍 HTML 的一些基本概念、常用元素，然后再介绍 CSS 的基本语法、CSS 选择器、常用的 CSS 属性和属性值。

2.1　HTML 基础

本节首先介绍 HTML 中的一些基本概念，然后再介绍一些常用 HTML 元素的使用。

2.1.1　HTML 文档

这里介绍 HTML 文档的基本结构，以及有关 HTML 标签和元素的概念。

1. 标签与元素

HTML 文档也称为网页，是一种文本文件，由要显示的内容和 HTML 标签（HTML tag）组成。HTML 标签用于指定文档的结构及相应的呈现格式。每个标签由一个小于号、标签名称和一个大于号构成，例如<p>。一般情况下，标签是成对出现的，即以开始标签起始、以结束标签结尾，两者之间的内容是标签的作用范围。与开始标签相比，结束标签的名称前多了一个斜杠（/），例如</p>。

通常，把从一个开始标签起始到其相应的结束标签终止的所有代码称为一个 HTML 元素（HTML element），而把开始标签与结束标签之间的内容称为该元素内容。有些元素没有内容，称为空元素。空元素不需要结束标签，通常在开始标签中关闭该元素，即在开始标签的尾部添加斜杠（/），例如
。

大多数 HTML 元素可拥有属性，用于提供有关 HTML 元素的更多的信息。属性总是在 HTML 元素的开始标签中指定，以名称-值对的形式出现，例如 name="user"，属性值可以用双引号或单引号括起来。

2. 基本结构

一个 HTML 文档总体上可分为头部和主体两部分，其代码类似如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>主页</title>
  </head>
  <body>
    <p>文档的内容</p>
  </body>
</html>
```

其中参数内容如下：

（1）<!DOCTYPE>声明必须是 HTML 文档的第一行，位于<html>标签之前。该声明并不是 HTML 标签，只是明确告诉 Web 浏览器这是一个 HTML 文档。对于有些浏览器来说，该声明是必不可少的。

（2）<html>与</html>标签限定了文档的开始点和结束点，在它们之间是文档的头部和主体。

（3）<head>元素用于定义文档的头部，它是所有头部元素的容器。文档的头部描述了文档的各种属性和信息，例如上述<title>元素定义文档的标题，<meta>元素指定文档所采用的字符集。除了标题会显示在浏览器窗口的标题栏上（或作为标签页的标签），文档头部定义的其他内容都不会显示在浏览器窗口中。

（4）<body>元素用于定义文档的主体。所有需要在浏览器窗口中显示的信息都应该定义在该元素内。

2.1.2 HTML 元素

本小节介绍 HTML 元素的分类以及元素之间的关系。

1. 元素分类

HTML 元素大致可分为块级元素和内联元素两类。

（1）块级元素（Block Element）。块级元素总是会在新的一行进行呈现。紧跟在块级元素后面的任何元素则会另起一行，在下一行进行呈现。

常见的块级元素包括 <h1>、<h2>、<h3>、<h4>、<h5>、<h6>、<p>、<div>、<hr>、<ul>、<ol>、<dl>、<li>、<table>、<form>等。

（2）内联元素（Inline Element）。内联元素并非需要从新的一行进行呈现。如果它前面的元素是块级元素，那么它自然会在新的一行进行呈现；否则，它会紧跟在前面的内联元素后面，在同一行中进行呈现。

常见的内联元素包括<a>、<img>、<span>、<label>、<input>、<textarea>、<select>、
、<sup>、<sub>等。

内联元素又可分为替换元素和非替换元素两类。

替换元素（Replaced Element）是指其内容来自外部而没有包含在文档中的一些元素，例如<img>、<input>、<textarea>、<select>等。替换元素往往是空元素，其内容由浏览器根据元素的标签和属性直接呈现，而不是由 CSS 模型负责呈现。

HTML 的大多数内联元素是非替换元素（Non-replaced Element），其内容包含在文档中，由 CSS 模型负责呈现。

2. 元素之间的关系

HTML 文档中元素之间存在包含和被包含的关系，大多数 HTML 元素都可以嵌入其他的 HTML 元素。

（1）父元素与子元素。包含另一个元素的元素是被包含元素的父元素，而被包含的元素是包含元素的子元素。如在上述 HTML 文档基本结构示例代码中，<html>元素是<head>和<body>元素的父元素，而<head>和<body>元素都是<html>元素的子元素。一个元素可以拥有多个子元素，但只能有一个父元素。

（2）后代元素与兄弟元素。包含在其他元素中的元素也可以包含别的元素。一个元素把其所包含的所有元素称为其后代元素，而子元素是指关系最近的后代元素。如上述 HTML 文档基本结构示例代码中，<body>和<p>元素都是<html>元素的后代元素，但是两者中只有<body>元素才是<html>元素的子元素。具有相同父元素的几个元素互为兄弟元素，如<head>和<body>元素是兄弟元素，因为它们的父元素都是<html>元素。

3. 全局属性

全局属性是指所有 HTML 元素都适用的属性，也称通用属性。下面是 HTML 元素常用的全局属性。

（1）id：为文档中的 HTML 元素指定独一无二的身份标识。

（2）class：为 HTML 元素指定的一个或多个类名（引用样式表中的类）。

（3）style：为 HTML 元素指定内联的 CSS 样式。

（4）title：为 HTML 元素指定标题信息。Web 浏览器通常会将该信息以即时提示的方式显示出来。

2.1.3 若干基本元素

这里介绍一些基本且常用的 HTML 元素。

1. 标题

HTML 元素<h1>、<h2>、<h3>、<h4>、<h5>和<h6>用于定义文档的各级标题，共 6 级，其中<h1>元素定义一级标题，字体最大；<h6>元素定义六级标题，字体最小。

【例 2-1】 HTML 标题元素的用法和效果如表 2-1 所示。

标题元素是一种块级元素。默认情况下，标题粗体显示，左对齐，各级标题的上下都有相应的外间距。

表 2-1 标题元素

HTML 代码	浏览器呈现
一级标题	一级标题
二级标题	二级标题
三级标题	三级标题

2. 段落

HTML 元素<p>用于定义文档的段落。段落元素是一种块级元素。默认情况下，每个段落的上下都有相应的外间距，但前后两个段落之间的间距会被重叠。

3. 换行

文本文件中的回车换行符（\r、\n 或\r\n）在浏览器中只是呈现为空格。为实现换行的效果，可以使用
元素。
元素使得任何紧跟在其后的内容另起一行呈现。

【例 2-2】 HTML 段落元素和换行元素的用法和效果如表 2-2 所示。

表 2-2 <p>和
元素

HTML 代码	浏览器呈现
这是一个段落， 但被强制换行了。	这是一个段落， 但被强制换行了。
这是另一个段落。	这是另一个段落。

元素是一种内联元素。
元素也是一种空元素，不需要结束标签，应在开始标签关闭该元素。

4. 水平线

HTML 元素<hr>可以呈现一条水平线，可用于分隔文档中不同部分的内容。

<hr>元素是一种块级元素。<hr>元素也是一种空元素，不需要结束标签，应在开始标签关闭该元素。

5. 与<div>

HTML 元素是内联元素，它没有特定的含义，通常用作文本的容器。通过设置该元素的 style 和 class 属性值，可以指定该元素的内容文本的呈现格式。

HTML 元素<div>是块级元素，它没有特定的含义，主要用作组合其他 HTML 元素的容器。通过设置该元素的 style 和 class 属性值，可以指定该元素块框的呈现格式。

【例 2-3】 、<div>和<hr>元素的用法和效果，如表 2-3 所示。其中，外层的块框通过 style 属性指定：宽度为 200 像素，边框为 1 个像素、黑色的实线。

6. 图像

<img>是非常有用的 HTML 元素，用于在网页中嵌入一幅图像，其 src 属性指定图像文件的 URL。如果图像文件与 HTML 文档处在同一个网站，一般采用相对地址。

<img>元素是一种替换内联元素。<img>元素也是一种空元素，不需要结束标签，应在开始标签关闭该元素。

表 2-3　、<div>和<hr>元素

HTML 代码	浏览器呈现
     文字 1文字 2 文字 3 	文字1文字2 文字3

【例 2-4】 HTML 图像元素的用法和效果，如表 2-4 所示。这里，图像文件位于页面文件所在目录的 image 子目录中，元素的 alt 属性指定当图像文件无法读取时可显示的替代文本，元素的 title 属性指定当鼠标指向图像时会显示的提示信息。style 属性指定 CSS 样式，设置图像的呈现大小。

表 2-4　HTML 图像元素

HTML 代码	浏览器呈现
	

7. 超链接

超链接是万维网的基本特征，一个 Web 页面通常会包含一些超链接。访问者通过单击超链接文字或图像，可以从一个页面跳转到另一个页面。这些页面可以处在同一个网站，也可以属于不同的网站。

HTML 元素<a>用于创建超链接，其 href 属性指定目标页面的 URL。如果目标页面与当前页面处在同一个网站，一般采用相对地址。

<a>元素是一种内联元素。开始标签和结束标签之间的元素内容会被呈现出来，当访问者单击它时，浏览器将自动请求 href 属性指定的目标页面。<a>元素的内容可以是普通文本，称为文本超链接；也可以<img>元素指定的图像，称为图像超链接。

【例 2-5】 HTML 超链接元素的用法和效果，如表 2-5 所示。默认情况下，超链接文字呈现时包含下画线，可以通过 CSS 样式重新进行设置。

表 2-5　HTML <a>元素

HTML 代码	浏览器呈现
Visit W3School	Visit W3School

8. 注释

可以在 HTML 文档中插入所需的注释，使 HTML 代码更易被人理解，可以提高文档的可读性。HTML 注释以“<!--”（4 个字符之间不要有空格）开始，以“-->”（3 个字符之间不要有空格）结束，可以横跨多行。例如：

```
<!-- 这是注释 -->
```

浏览器会忽略 HTML 注释，也不会显示它们。

2.2 列　　表

HTML 列表包括无序列表、有序列表和定义列表。这里，各种列表元素都是块级元素，而且各种列表中的列表项元素也是块级元素。

2.2.1 无序列表

无序列表是一种项目列表，其中每个列表项前会显示一个项目符号（默认为实心圆点）。无序列表始于<ul>标签，其中每个列表项始于<li>标签。

【例 2-6】 无序列表的用法和效果，如表 2-6 所示。

表 2-6　无序列表（<ul>和<li>）

HTML 代码	浏览器呈现
  - 图像 - 超链接 	• 图像 • 超链接

项目符号可以由 CSS 属性 list-style-type 设置，其取值如：

- disc（默认值，圆点）；
- circle（小圆圈）；
- square（小正方形）；
- none（无）。

列表项内部可以使用段落、换行、图片、超链接等元素，也可包含其他列表。

2.2.2 有序列表

有序列表是一种编号列表，其中每个列表项前会使用数字或字母进行标记（默认为阿拉伯数字）。有序列表始于<ol>标签，其中每个列表项始于<li>标签。

【例 2-7】 有序列表的用法和效果，如表 2-7 所示。

表 2-7　有序列表（<ol>和<li>）

HTML 代码	浏览器显示
  - 图像 - 超链接 	1. 图像 2. 超链接

列表项前的编号也可以由 CSS 属性 list-style-type 设置，其取值如下：

- decimal（默认值，数字）；
- lower-alpha（小写字母）；

- upper-alpha（大写字母）；
- lower-roman（小写罗马数字）；
- upper-roman（大写罗马数字）；
- none（无）。

列表项内部可以使用段落、换行、图片、超链接等元素，也可包含其他列表。

2.2.3 定义列表

定义列表中的每个列表项是其自身与其定义的组合。定义列表以<dl>标签开始，其中每个列表项以<dt>开始，而每个列表项的定义以<dd>开始。

【例 2-8】 定义列表的用法和效果，如表 2-8 所示。其中用到了 HTML 实体“<”（表示符号“<”）和“>”（表示符号“>”）。

表 2-8 定义列表（<dl>、<dt>和<dd>）

HTML 代码	浏览器呈现
<dl> <dt>元素<img></dt> <dd>创建图像，src 指定图像。</dd> <dt>元素<a></dt> <dd>创建超链接，href 指定目标页面。</dd> </dl>	元素<img> 创建图像，src指定图像。 元素<a> 创建超链接，href指定目标页面。

列表项内部可以使用段落、换行、图片、超链接等元素，也可包含其他列表。

2.3 表　　格

HTML 表格由<table>元素及其子元素定义。常用的子元素包括<tr>、<th>、<td>等，其中 tr 元素定义表格行，th 元素定义表头单元格，td 元素定义数据单元格。

复杂的 HTML 表格还可能包括<caption>、<col>、<colgroup>、<thead>、<tfoot>以及<tbody> 等子元素。

<table>元素是一种块级元素。与其他块级元素不同的是，表格的默认宽度不是容器的宽度，而是由其内容决定的。

本小节介绍 HTML 表格的定义，也会涉及一些表格格式的设置。

2.3.1 简单的表格

表格由<table>标签来定义。表格中的行由<tr>标签定义，行中的单元格由<td>或<th>标签定义。每个单元格内可以呈现文本、图像、超链接等内容，甚至可以包含另外的表格。通常，<td>用来定义数据单元格，<th>用来定义表头单元格。默认情况下，<td>定义的单元格，其内容水平左对齐、垂直居中对齐；<th>定义的单元格，其内容水平居中、垂直居中，文字粗体显示。

【例 2-9】 一个简单表格的定义，如表 2-9 所示。<table>元素的 border 属性指定表格边

框线的粗细（以像素为单位）。默认情况下，如果指定 border 属性值为非零，那么除了显示指定粗细的表格边框线，也会显示各单元格的边框线。

表 2-9　定义简单的表格

HTML 代码

```
<table border="1">
    <tr>
        <th>姓名</th><th>性别</th><th>部门</th>
    </tr>
    <tr>
        <td>周小兰</td><td>女</td><td>会计学院</td>
    </tr>
    <tr>
        <td>胡之军</td><td>男</td><td>信息学院</td>
    </tr>
</table>
```

浏览器呈现

姓名	性别	部门
周小兰	女	会计学院
胡之军	男	信息学院

一般情况下，在 HTML 元素中，用于表示大小、长度、粗细等的属性，其值都以像素为单位（不需要写出）。

2.3.2　跨行与跨列

colspan 和 rowspan 是<td>和<th>标签的属性。colspan 属性用于指定单元格横跨的列数；rowspan 属性用于指定单元格占据的行数。

【例 2-10】 colspan 和 rowspan 属性的用法和效果，如表 2-10 所示。

表 2-10　跨行（rowspan）和跨列（colspan）

HTML 代码

```
<table border="1">
    <tr>
        <th>Column 1</th>
        <th>Column 2</th>
        <th>Column 3</th>
    </tr>
    <tr>
        <td rowspan="2">Row 1 Cell 1</td>
        <td>Row 1 Cell 2</td>
        <td>Row 1 Cell 3</td>
    </tr>
    <tr>
        <td>Row 2 Cell 2</td>
        <td>Row 2 Cell 3</td>
    </tr>
    <tr>
        <td>Row 3 Cell 1</td>
        <td colspan="2">Row 3 Cell 2</td>
    </tr>
</table>
```

浏览器呈现

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1	Row 3 Cell 2	

2.3.3 标题、表头、表体和表脚

<caption>元素定义表格标题，标题置于元素开始标签和结束标签之间。<caption>元素内可以包含除<table>元素之外的任何元素。

<caption>元素应该紧随<table>元素之后。每个表格只能定义一个标题。通常这个标题会被居中于表格之上。

除了标题，一个表格可以被划分为 3 个分区：表头、表体和表脚。表头可以用<thead>元素定义，属于表头的<tr>元素应该放置在<thead>元素内。表体可以用<tbody>元素定义，属于表体的<tr>元素应该放置在<tbody>元素内，一个表格可以包含多个<tbody>元素。表脚可以用<tfoot>元素定义，属于表脚的<tr>元素应该放置在<tfoot>元素内。

作为<table>元素的子元素，它们在<table>元素内的出现的先后次序通常是<caption>、<thead>、<tfoot>和<tbody>。

【例 2-11】 定义表格的标题、表头、表体和表脚，如表 2-11 所示。

表 2-11 定义表格的标题、表头、表体和表脚

HTML 代码

```
<table border="1">
    <caption>This is the title of table</caption>
    <thead>
        <tr>
            <th>head1</th>
            <th>head2</th>
            <th>head3</th>
        </tr>
    </thead>
    <tfoot>
        <tr>
            <td colspan="3">
                This is the foot of the table
            </td>
        </tr>
    </tfoot>
    <tbody>
        <tr>
            <td>Row1 Cell 1</td>
            <td>Row1 Cell 2</td>
            <td>Row1 Cell 3</td>
        </tr>
        <tr>
            <td>Row2 Cell 1</td>
            <td>Row2 Cell 2</td>
            <td>Row2 Cell 3</td>
        </tr>
    </tbody>
</table>
```

浏览器呈现

This is the title of table

head1	head2	head3
Row1 Cell 1	Row1 Cell 2	Row1 Cell 3
Row2 Cell 1	Row2 Cell 2	Row2 Cell 3
This is the foot of the table		

2.3.4 边框与单元格间距

利用<table>元素的 border、frame 和 rules 等属性，可以控制是否显示表格及其单元格的边框线，以及表格边框线的粗细。为了更加灵活地选择边框的样式、粗细、颜色，可以利用 CSS 的 border 属性对表格和单元格的边框分别进行设置。

cellspacing 是<table>元素的属性，用于指定单元格与单元格以及单元格与表格边框之间的间距，即单元格的外间距。cellpadding 也是<table>元素的属性，用于指定单元格的内容与单元格边框之间的间距，即单元格的内间距。

CSS 属性 border-collapse 用于控制是否去除单元格之间及单元格和表格边框之间存在的空间。默认情况下，或该属性设置为 separate 时，空间不被去除，相邻边框线是分离的；如果将该属性设置为 collapse，那么上述空间将被去除，并且相邻边框线合二为一。

【例 2-12】 使用 CSS 样式控制表格及其行（或单元格）的边框线的显示，如表 2-12 所示。其中，将单元格的外间距设置为 0，单元格的内间距设置为 10px。另外，将表格上、下边框设置为 1 个像素的实线，将表头行的下边框设置为 2 个像素的实线。

表 2-12 显示表格及其行的边框线

HTML 代码	浏览器呈现
<table cellpadding="10" style="border-collapse: collapse; border-bottom: 1px solid black; border-top: 1px solid black"> <tr style="border-bottom: 2px solid black"> <th>姓名</th><th>性别</th><th>部门</th> </tr> <tr> <td>周小兰</td><td>女</td><td>会计学院</td> </tr> <tr> <td>胡之军</td><td>男</td><td>信息学院</td> </tr> </table>	**姓名 性别 部门** 周小兰 女 会计学院 胡之军 男 信息学院

注意：只有当 CSS 属性 border-collapse 被设置为 collapse 时，才可以通过 CSS 属性 border 有效设置表格行 tr 的边框。

2.3.5 为列指定 CSS 样式

<col>元素为表格中的一个或多个列定义 CSS 样式。该元素的 span 属性指定列数，style 和 class 属性指定 CSS 样式。

<col>元素是仅包含属性的空元素，它本身并不定义列内容，只是为列定义 CSS 样式。该元素只能出现在<table>或<colgroup>元素中。

<colgroup>元素具有与<col>元素相似的功能和用法，即也可以为表格中的一个或多个列定义 CSS 样式。除此之外，<colgroup>可以作为<col>元素的父元素，这样就可以通过<colgroup>元素为其中的各<col>元素指定的所有列定义共同的 CSS 样式。

需要注意的是，仅有以下 CSS 样式属性可以通过<col>和<colgroup>元素应用于表格列。

- border：为列指定边框，仅在表格的 border-collapse 属性设置为 collapse 时有效。
- background：为列中各单元格设置背景颜色或背景图案，仅在单元格及其所在行有透明背景时有效。
- width、min-width：指定列的宽度和最小宽度。
- visibility：当该属性被设置为 collapse 时，该列不被呈现。

作为<table>元素的子元素，它们在<table>元素内出现的先后次序通常是<caption>、<col>（<colgroup>）、<thead>、<tfoot>和<tbody>。

【例 2-13】 <col>元素的用法和效果，如表 2-13 所示。其中，第 1 列的背景颜色设置为浅灰色，第 3 列设置了右边框。

表 2-13　利用<col>元素设置表格列的格式

HTML 代码

```
<table cellpadding="10"
        style="border-collapse: collapse;
        border-bottom: 1px solid black">
    <col style="background-color: lightgray" />
    <col />
    <col style="border-right: 1px solid black" />
    <tr style="color: white; background-color: darkgray">
        <th>姓名</th><th>性别</th><th>部门</th>
    </tr>
    <tr>
        <td>周小兰</td><td>女</td><td>会计学院</td>
    </tr>
    <tr>
        <td>胡之军</td><td>男</td><td>信息学院</td>
    </tr>
</table>
```

浏览器呈现

姓名	性别	部门
周小兰	女	会计学院
胡之军	男	信息学院

2.4　表　　单

HTML 表单用于收集用户输入的数据，并将这些数据送往 Web 服务器供指定程序处理。表单由<form>元素定义，该元素内通常会包含若干表单控件元素，不同的类型的表单控件元素会以不同的方式接收用户的输入。

2.4.1 表单元素<form>

HTML 表单由<form>元素定义。其一般格式如下：

```
<form>
  <表单控件元素>…
</form>
```

<form>元素通常需要指定某些属性。<form>元素的属性如下。

（1）action。指定由哪个后台程序处理用户的输入数据。默认值是当前页面。

（2）method。指定请求方法，决定以什么方式向服务器传递用户数据。属性值可取 GET 或 POST，默认值为_self（当前窗口或框架）。

（3）target。指定呈现后台程序处理结果的窗口或框架，如_blank, _self, _parent 等。默认值是当前窗口。

（4）enctype。指定浏览器应如何编码要发送的数据。可能的值如下所示：

- application/x-www-form-urlencoded（默认值）
- multipart/form-data（上传文件时使用）

2.4.2 <input>元素

<input>是最主要的表单控件元素，其 type 属性指定控件的具体类型。下面介绍<input>元素的一些属性。

（1）type：设置控件类型。

- text：单行文本域。
- password：密码域。
- checkbox：复选框（检测框）。
- radio：单选按钮。
- file：文件域。
- submit：提交按钮。
- reset：重置按钮。
- button：自定义按钮。
- image：图像提交按钮。

（2）name：指定控件的名称。当提交表单时，表单各控件的值以请求参数形式送往 Web 服务器，参数的名称就是 name 属性的值。一组单选按钮的 name 属性值应该取相同值，这样用户只能在其中选择一个单选按钮。

（3）value：指定控件的初始值。

（4）maxlength：指定用户能在控件中输入的最大字符数。适用于文本域和密码域。

（5）size：以字符为单位的控件宽度。适用于文本域和密码域。

（6）src：如果 type 属性指定为 image，那么该属性指定图像的 URL。

（7）checked：指定控件处于选中状态，适用于检测框和单选按钮。例如 checked="checked"。

<input>元素是一种替换内联元素。<input>元素也是一种空元素，不需要结束标签，应在开始标签关闭该元素。

【例 2-14】 文本域和密码域的定义及效果，如表 2-14 所示。这是一个简单的登录表单，考虑密码的安全性，采用 POST 请求方法。当用户单击“登录”按钮时，将有 3 个参数伴随着请求送往服务器端：名为 user 的参数，其值为用户输入的用户名；名为 pw 的参数，其值为用户输入的密码；名为 login 的参数，其值为“登录”。

表 2-14　文本域与密码域

HTML 代码	浏览器呈现
<form method="post"> 用户名：<input type="text" name="user" value="" /> 密码：<input type="password" name="pw" value="" /> <input type="submit" name="login" value="登录" /> </form>	用户名：[loubuye] 密码：[••••••] [登录]

2.4.3　为控件元素指定标签

可以使用<label>元素为控件元素指定标签，使该标签与控件元素建立关联。当用户单击标签时，相关联的控件将会获得焦点，从而可以提高用户的可操作性。

<label>元素的 for 属性值应该设置为相关联的控件元素的 id 属性值。

【例 2-15】 标签的使用，如表 2-15 所示。这里分别为一个文本域和两个单选按钮定义了标签。当用户单击某个标签时，相当于单击了对应的控件。

表 2-15　使用<label>元素

HTML 代码	浏览器呈现
     <label for="i1">用户名：</label> <input id="i1" type="text" name="user" value="" /> 性别： <input id="a" type="radio" name="gender" value="m" /> <label for="a">男</label> <input id="b" type="radio" name="gender" value="f" /> <label for="b">女</label> 	用户名：[　　　　] 性别：○ 男　○ 女

2.4.4　<textarea>元素

<textarea>元素定义文本区，允许用户输入多行文本。文本区控件没有 value 属性，其初值写在开始标签和结束标签之间，即元素的内容作为初值。<textarea>元素是一种替换内

联元素。下面是<textarea>元素常用的一些属性。

（1）name：指定文本区控件的名称。

（2）maxlength：指定用户能在文本区输入的最大字符数。

（3）rows：以行为单位的文本区的高度。

（4）col：以列为单位的文本区的宽度。

（5）readonly：指定文本区是只读的，例如 readonly="readonly"。

（6）disabled：指定文本区是禁用的，例如 disabled="disabled"。

只读控件和禁用控件的值都是不能修改的，但它们有明显的区别：只读控件是可聚焦的（能获得光标），而禁用控件不能聚焦；只读控件的值会被提交，而禁用控件的值不被提交。

【例 2-16】 文本区的定义，如表 2-16 所示。这里，文本区与其标签“说明”在同一行。为了让标签与文本区的顶端对齐，对该标签设置了相应的 CSS 样式属性，即"vertical-align: top"。

表 2-16　使用<textarea>元素

HTML 代码	浏览器呈现
 <label for="i1">名称：</label> <input id="i1" type="text" name="pid" value="" /> <label for="i2" style="vertical-align: top">说明：</label> <textarea id="i2" name="specifier" rows="5" cols="30"> </textarea> 	名称： 说明：

2.4.5　选择列表

选择列表允许用户从一组预定义的选项中进行选择。HTML 选择列表是一种表单控件元素，由<select>元素定义，放置在<form>元素内。选择列表中的每个选项由<option>元素定义，放置在<select>元素内。

除了全局属性，<select>元素还经常使用以下属性。

（1）name：指定选择列表控件的名称。

（2）size：指定选择列表中可见选项的数目。默认情况下，size 属性值为 1，此时选择列表呈现为下拉菜单。若将 size 设置为>1，则选择列表呈现为列表框。

（3）multiple：指定可选择多个选项，例如 multiple="multiple"。在多选情况下，选择列表一般呈现为列表框。

（4）disabled：指定选项列表是禁用的，例如 disabled="disabled"。

每个<option>元素定义一个选项。选项的标签（显示文本）通常作为元素的内容出现在元素的开始标签和结束标签之间。<option>元素还经常使用以下属性。

（1）value：指定选项的值。

（2）selected：指定选项是预选择的。如果没有任何选项设置为预选择的，那么对于单选来说，第一个选项默认是预选择的；对于多选来说，没有选项是预选择的。

（3）disabled：指定选项是禁用的。

2.5 初识 CSS

CSS（Cascading StyleSheets，层叠样式表）是一种用于指定网页内容呈现格式的计算机语言。使用 CSS 技术的优点如下。

（1）网页的文档内容与其呈现格式的定义的分离，可以提高网页代码的可读性。

（2）对网站所有或部分网页的呈现格式进行统一的定义，可以确保网站具有一致的呈现风格。

（3）一些呈现样式通常会被一个或多个网页反复使用，因此可以减少页面的代码数量，提高下载速度。

（4）对网站所有或部分网页的呈现格式进行集中定义，便于修改和维护，可以降低网站的开发和维护工作量。

网页内容的呈现格式由相关样式表中的规则（样式）定义。一个 CSS 样式表由若干规则（样式）组成，每个规则由选择器和声明块两部分组成。声明块起始于左花括号（{）、终止于右花括号（}），内含若干声明，每个声明是某个 CSS 属性的名称-值对。定义规则（样式）的一般格式如下：

```
<选择器> {<属性名>:<属性值> [;<属性名>:<属性值>]*}
```

其中，选择器指定相关声明作用的 HTML 元素。属性名与属性值之间用冒号（:）分隔，各属性名-值对之间用分号（;）分隔，最后一个分号可以省略。这里，各语法符号（{、}、:、;）前后都可以有不定数量的空白符号。

为了表示某个声明的重要性，可以在声明后紧跟!important，称为!important 声明。例如，如下是一个规则定义：

```
p {font-size: 12px; color: red!important}
```

其中，p 是选择器，表明该规则作用于段落元素。声明块中包含两个声明，其中第 2 个声明是一个!important 声明。

在定义 CSS 样式表时，可以使用注释。CSS 注释以“/*”开始，以“*/”结束。注释可以是单行的，也可以是多行的，可以出现在 CSS 代码的任何地方。

2.6 CSS 选择器

在 CSS 中，选择器大致可分为基本选择器、层次选择器、伪类选择器和伪元素选择器 4 种，下面依次介绍。

2.6.1 基本选择器

基本选择器由元素选择器或通用选择器紧跟零个或多个次序无关的类选择器、属性选择器、ID 选择器和伪类选择器组成，各选择器之间不含空格。如果是通用选择器紧跟其他选择器的形式，通用选择器可省略。

1. 元素选择器

元素选择器用 HTML 元素的名称作为选择器，又称类型选择器，用于匹配所有由选择器指定的特定类型元素。元素选择器的格式如下：

```
<元素名>
```

例如，下面规则指定所有三级标题文字用蓝色、黑体显示。

```
h3 {color:blue; font-family:黑体}
```

2. 通用选择器

通用选择器用星号（*）作为选择器，用于匹配所有元素。通用选择器可看作是元素选择器的一个特例。

例如，下面规则指定所有元素的外边距为 0，内边距为 0。

```
* {margin:0; padding:0}
```

3. 类选择器

类选择器由点号（.）紧跟一个自定义的类名来表示，用于匹配所有 class 属性值包含指定类名的元素。类选择器其格式如下：

```
.<类名>
```

例如，如下规则定义：

```
.c1 {font-size:12px; letter-spacing:2px}
p.c2 {width:90%; background-color:blue}
```

第 1 条规则指定所有 class 属性值包含 c1 的元素，呈现时的字体大小为 12 个像素、字间距为 2 个像素。第 2 条规则指定所有 class 属性值包含 c2 的<p>元素，呈现时的段落宽度为容器宽度的 90%、背景颜色为蓝色。

【例 2-17】 使用类选择器。一个元素的 class 属性可以包含多个类名，各类名之间用空格分隔，次序无关紧要。根据定义的 CSS 规则，说明下面 HTML 文档中各段落的呈现格式：

```
<head>
<style>
   .error {color: red;}
   .warning {font-style:italic;}
   .error.warning {background:silver;}
</style>
</head>
```

```
<body>
    <p class="error">This is an error.</p>
    <p class="warning">This is a warning.</p>
    <p class="error warning">This is a warning and error.</p>
</body>
```

解答：这里，样式表中定义了 3 条规则，其中第 3 条规则指定，若一个元素的 class 属性值既包含类名 error 又包含类名 warning，那么元素以银灰色作为背景。

文档呈现的效果为：第 1 个段落文本以红色显示；第 2 个段落文本以斜体显示；第 3 个段落文本以红色、斜体显示，背景颜色为银灰色。

4. ID 选择器

ID 选择器由#号紧跟一个自定义的 id 值组成，用于匹配页面中 id 属性值为指定 id 值的 HTML 元素。ID 选择器格式如下：

```
#<id 值>
```

例如，如下规则定义：

```
#e {font-weight: bold}
span#e {color: red}
```

第 1 条规则指定 id 属性值为 e 的元素的内容将以粗体显示。第 2 条规则指定 id 属性值为 e 的<span>元素的内容将以红色显示。

5. 属性选择器

属性选择器用方括号（[ ]）来表示，用于匹配所有包含某属性或某属性具有某值的元素。属性选择器的格式如下：

```
[<属性名>]
```

或

```
[<属性名>=<属性值>]
```

例如，如下规则定义：

```
[href] {color: blue}
input[type=password] {background-color: gray}
```

第 1 条规则规定网页中所有指定了 href 属性的元素内容以蓝色显示。第 2 条规则规定所有指定了 type 属性且其值为 password 的<input>元素（口令域）将以灰色背景显示。

又比如，如下规则定义：

```
span[title].cc {color: blue}
```

或

```
span.cc[title] {color: blue}
```

这两个规则具有相同的效果，指定所有带有 title 属性且 class 属性值包含 cc 的<span>元素内

容以蓝色显示。

6. 分组选择器

当某些规则具有相同的声明时，可以把这些规则的选择器用逗号（,）分隔，形成分组选择器，而声明只需写一次即可。

例如，如下规则：

```
h1 {color: green}
h2 {color: green}
h3 {color: green}
```

等价于

```
h1, h2, h3 {color: green}
```

分组选择器匹配组中任何一个选择器匹配的元素。如上述规则中的声明应用于所有一级标题、二级标题和三级标题。

2.6.2 层次选择器

层次选择器由层次符将若干基本选择器连接起来形成，层次符前后可以有空白符号。层次符包括空白符号、大于号（>）和加号（+），分别表示后代、子代和相邻兄弟。

层次选择器的匹配过程如下：首先从页面中找出与最左边基本选择器相匹配的元素；然后基于这个匹配结果，根据其后的层次符的含义，确定一个元素集合；接着在这个元素集合中找出与下一个基本选择器相匹配的元素；这个匹配过程依次进行，直至最后一个基本选择器，最后一个基本选择器匹配的元素就是层次选择器匹配的对象。

1. 后代选择器

后代选择器通过空白符号将两个基本选择器连接起来形成，用于匹配包含在某些元素（与基本选择器 1 相匹配）中的某些后代元素（与基本选择器 2 相匹配）。后代选择器的格式如下：

```
<基本选择器 1> <基本选择器 2>
```

例如，如下规则定义：

```
div#sidebar {background-color: blue;}
div#sidebar a[href] {color: white;}
```

第 1 条规则指定 id 属性值为 sidebar 的<div>元素采用蓝色背景。第 2 条规则指定上述<div>元素中所有包含 href 属性的<a>元素以白色作为前景颜色。

2. 子代选择器

子代选择器通过大于号（>）将两个基本选择器连接起来形成，用于匹配包含在某些元素（与基本选择器 1 相匹配）中的某些子元素（与基本选择器 2 相匹配）。子代选择器的格式如下：

```
<基本选择器 1> > <基本选择器 2>
```

例如，如下规则定义：

```
div#main > .title {font-weight: bold}
body > * p {color: red;}
```

第 1 条规则指定 id 属性值为 main 的<div>元素的所有 class 属性值包含 title 的子元素用粗体显示。第 2 条规则指定<body>元素中所有非子代的后代<p>元素以红色显示。

3. 相邻兄弟选择器

相邻兄弟选择器通过加号（+）将两个基本选择器连接起来形成，用于匹配某些元素（与基本选择器 1 相匹配）后面与基本选择器 2 相匹配的某些相邻兄弟元素。相邻兄弟选择器的格式如下：

```
<基本选择器 1> + <基本选择器 2>
```

如果与基本选择器 1 相匹配的元素后面没有相邻兄弟元素，或相邻兄弟元素与基本选择器 2 不匹配，那么相邻兄弟选择器不匹配任何元素。

例如，如下规则定义：

```
ul > li + li {color: blue}
```

该规则指定，在所有无序列表中，除第 1 个列表项，其他各列表项均以蓝色显示。

2.6.3 伪类选择器

一般选择器基于元素名、id 属性、class 属性或其他普通属性匹配元素。与此不同，伪类选择器基于元素的其他一些特征来匹配元素。伪类选择器用冒号（:）紧跟一个伪类名来表示，其格式如下：

```
:<伪类名>
```

与属性选择器等一样，伪类选择器属于基本选择器的范畴，也可以出现在层次选择器的任何部分。

下面介绍一组常用的伪类选择器，它们根据动态状态来匹配元素，这些状态因用户与页面交互产生，并不是页面文档本身的一部分。

1. :link 和 :visited

通常，浏览器会用不同的样式呈现未被访问过的超链接和已被访问过的超链接。用户也可以通过伪类选择器:link 和:visited，对这两种状态的样式进行重新定义。

（1）:link：匹配未被访问过的超链接元素。

（2）:visited：匹配已被访问过的超链接元素。

由于这两个伪类选择器仅适用于超链接元素<a>，所以选择器:link 和 a:link 是等效的，选择器:visited 和 a:visited 也是等效的。

2. :hover、:active 和 :focus

通过这些伪类选择器为元素的不同状态定义不同的样式，可以提高用户与页面交互的友好程度。

（1）:hover：匹配用户鼠标悬停在其上的元素。鼠标在页面上移动时，掠过的元素可以用特定的样式呈现。

（2）:active：匹配当前用户激活的元素。这里，激活通常是指鼠标键被按下与释放之间的状态。

（3）:focus：匹配当前获得焦点的元素。能获得焦点的元素主要是指表单控件元素。

例如，如下规则定义：

```
input[type=text]:focus,  input[type=password]:focus  {background-color:
whitesmoke}
```

该规则指定，文本域和密码域控件聚焦时会以白雾色作为背景颜色。

又比如，以下规则定义：

```
a:link {color: steelblue; text-decoration: none}
a:visited {color: steelblue; text-decoration: none}
a:hover {color: red; text-decoration: underline}
a:active {font-size: smaller; text-decoration: underline}
```

这组规则规定，未被访问过和已被访问过的超链接都以钢青色显示且没有下画线；鼠标悬停时的超链接将以红色显示且有下画线；激活时的超链接将以小号字显示且有下画线。

需要注意的是，上面针对超链接不同状态的规则定义，顺序是不能改变的。因为有些状态是重叠的，比如，当鼠标悬停在某超链接元素上时，该元素首先是一个未被访问或已被访问的超链接。又比如，当用户单击一个超链接元素时，鼠标同时也悬停在该超链接元素之上。如果改变上述定义的顺序，有些效果就可能出不来了。

2.6.4　伪元素选择器

伪元素选择器并非直接对应 HTML 文档中定义的元素，它为开发者提供了一种途径，可以对文档元素的部分内容指定样式，或者在某元素前后添加内容并设置样式。

与伪类选择器不同，伪元素选择器只能出现在选择器的末尾，比如对于层次选择器，伪元素选择器只能出现在最后一个基本选择器的末尾。

1．:first-line

伪元素选择器:first-line 匹配文本块的首行。首先，它匹配的对象只能是包含文本的块级元素。其次，它只应用于首行内容，如果因浏览器窗口缩放导致首行内容变化，那么选择器匹配的对象内容也随之变化。如果文本块的首行没有文本，则无法匹配。

例如，如下规则定义：

```
:first-line {font-weight: bold}
p:first-line {color: red}
```

第 1 条规则指定，所有块级元素的首行文字以粗体显示。第 2 条规则指定，所有<p>元素的首行文字以红色显示。

2．:first-letter

伪元素选择器:first-letter 匹配文本块的首行首字符。如果文本块首行的行首不是字符，

则无法匹配。

例如，如下规则定义：

```
p:first-letter {font-size: 2em}
```

该规则指定，所有段落的首行首字符以当前字号的两倍呈现。

3．:before 和:after

伪元素选择器:before 可以在元素的内容之前插入内容。伪元素选择器:after 可以在元素的内容之后插入内容。在为这两个伪元素选择器指定声明时，一般都应通过 content 属性指定要插入的内容，另外还可以用其他属性为插入内容指定样式。

例如，如下规则定义：

```
a:before {content: "Click here to "; font-style: italic}
a:after {content: "!"}
```

第 1 条规则指定，所有<a>元素的内容（超链接文字）前面都加上文字"Click here to "，并以斜体显示。第 2 条规则指定，所有<a>元素的内容（超链接文字）后面都加上文字"!"。

2.7 使用 CSS

前面介绍了 CSS 的基本语法及如何指定选择器等，本节将介绍 CSS 样式表的几种存在方式以及如何将其应用于页面及其元素。

2.7.1 定义和使用样式表

样式表分内部样式表和外部样式表两种。除此之外，还经常使用内联样式。

1. 内联样式

内联样式是指在 HTML 元素的 style 属性中指定的若干声明，即一个或多个 CSS 属性的名称-值对。

内联样式是脱离规则和样式表而直接在页面元素中指定的声明。内联样式仅作用于所在的元素。

内联样式用法简单，其应用效果也很容易观察，但没有将其与要应用的 HTML 元素分离，会使其失去了 CSS 技术本身的一些优势。通常，内联样式可用于那些需要特殊样式的 HTML 元素，为这些元素声明特定的 CSS 属性。

2. 内部样式表

内部样式表定义于页面内，其样式可应用于页面内的 HTML 元素。内部样式表在 style 元素内定义，而 style 元素一般放置在页面的 head 元素内。下面代码演示了定义内部样式表的一般格式。

```
<head>
  <style type="text/css">
  <!--
    p {font-size:12px; color:steelblue}
```

```
    .cone {font-family:楷体_gb2312; text-align:center}
  -->
  </style>
</head>
```

这里，把整个样式表包含在 HTML 注释内，可以避免不支持 CSS 的浏览器把样式表内容直接显示出来。而对于支持 CSS 的浏览器，则会对其进行分析，并将其中的规则应用于页面中的相关元素。由于 CSS 已成为一种标准，一般的浏览器都应该支持 CSS，所以通常也可以省略其中的 HTML 注释标签。

3. 链接外部样式表

内联样式和内部样式表都不能很好地体现 CSS 技术的优势。一个定义好的规则，不能仅仅能用于一个元素或一个页面，而是应该能用于所有需要它的页面和元素。

可以将一个样式表定义保存在一个单独的文件中，称为样式表文件。样式表文件是一个文本文件，其扩展名应该为.css。

在页面中可以用 link 标签链接一个样式表文件。link 标签用于在当前 HTML 页面与样式表文件（或其他外部文档）之间建立链接，使得页面中的元素可以使用被链接的样式表中的样式。与定义内部样式表一样，link 标签一般也应该写在页面的 head 元素内。下面代码演示了链接外部样式表的方法：

```
<head>
  <link rel="stylesheet" type="text/css" href="css/cssone.css"/>
</head>
```

其中，rel 属性指定当前页面文档与被链接外部文档之间的关系，在链接样式表文件时，通常取值为 stylesheet；type 属性指定被链接外部文档的 MIME 类型，对于样式表文件，其值总是 text/css；href 属性用于指定外部样式表文件的地址。在上面代码中，被链接的样式表文件 cssone.css 应该存放在当前页面所在目录的 css 子目录中。

一个样式表文件可以被多个页面文档链接。反过来，一个页面也可以链接多个样式表文件。一个页面除了可以链接外部样式表，还可以同时用 style 标签定义内部样式表。但需要注意，这些样式表的链入或定义的先后次序会影响其中的样式的优先级。

说明：可以将 rel 属性指定为 alternate stylesheet，并通过 title 属性指定一个标题。这样，被链接的外部样式表就成为一个可替换的样式表。页面的读者可以通过指定的标题来选择要应用于页面的样式表。Firefox、Opera 等浏览器支持这种交互功能。

4. 导入外部样式表

导入外部样式表是指用@import 语句在一个样式表中链接另一个样式表。也就是说一个样式表除了能定义自己的样式外，还可以包含另一个样式表的样式，但@import 语句必须出现在其他样式定义之前。下面代码演示了导入外部样式表的方法。

```
<head>
  <style type="text/css">
    <!--
    @import "css/cssone.css";
```

```
        @import "css/csstwo.css";
        p {font-size:12px; color:steelblue}
        -->
    </style>
</head>
```

这个代码在内部样式表中导入 url 地址分别为“css/cssone.css”和“css/csstwo.css”的两个样式表。除了可用于内部样式表，@import 符号也可用于外部样式表，即在一个外部样式表中导入另外的外部样式表。

说明：link 是一个 HTML 标签，用于在 HTML 页面中链接一个外部样式表或其他类型的外部文档。@import 是一个 CSS 语句，用于在一个样式表中导入另外的外部样式表。

2.7.2　层叠处理

一个页面的呈现可能会用到若干样式表，而每个样式表又会包含很多规则和声明。这样就很可能会导致以下冲突现象：有多个声明可应用于同一个元素，而这些声明又具有相同的属性名。这时，通常需要 CSS 从中选取优先级最高的声明应用于该元素，这个过程称为层叠。

1．!important 声明

在定义规则时，可以在声明后紧跟!important 以表示该声明的重要性，称为!important 声明。在层叠处理中，!important 声明总是比普通声明具有更高的优先级。

2. 样式表的来源

前面，从作者（即页面开发者）的角度介绍了样式表的定义和使用。除此之外，页面在呈现时还会用到另外两种样式表：

（1）用户代理（浏览器）的默认样式表。HTML 元素的作用主要是表示文档的结构，但默认情况下也会呈现出与文档结构相适应的格式，如每个段落前后会有一个外间距。实际上，这种默认格式产生于用户代理的默认样式表。

（2）用户样式表。除了浏览器的默认样式表和作者定义的样式表，用户（即页面浏览者）也可以定义自己的样式表。用户样式表的定义过程通常依赖于浏览器的类型，这里不作具体介绍。

来源不同的样式表具有不同的权重，通常作者样式表的权重比用户样式表的权重大，用户样式表的权重比用户代理的默认样式表的权重大。在层叠处理中，权重大的样式表中的规则和声明具有更高的优先级。

作为一种平衡，来自用户样式表的!important 声明比来自作者样式表的!important 声明的优先级要高。

综合声明的重要性和样式表的来源两方面的因素，层叠处理采用以下从高到低的优先级次序。

（1）用户样式表中的!important 声明；

（2）作者样式表中的!important 声明；

（3）作者样式表中的规则和声明；

（4）用户样式表中的规则和声明；

（5）用户代理（浏览器）默认样式表中的规则和声明。

3. 选择器的特指性

规则的选择器用于指定相关声明可应用的元素。有些选择器匹配的元素是比较具体的，特指性比较强，比如 ID 选择器通常仅能匹配页面中的一个元素，因为一个页面中各元素的 id 属性值应该是唯一的。有些选择器匹配的元素是比较宽泛的，特指性比较弱，比如元素选择器往往能匹配多个元素，因为一个页面通常会包含很多同类型的元素。

一个选择器通常由元素选择器、ID 选择器、类选择器、属性选择器等子选择器组成的。一个选择器的特指性计算如下：

（1）若声明定义于元素的 style 属性（内联样式，没有选择器），则 a=1，b=0，c=0，d=0；否则 a=0。

（2）将 b 设置为 ID 子选择器出现的次数。

（3）将 c 设置为类子选择器、属性子选择器和伪类子选择器出现的次数。

（4）将 d 设置为元素子选择器和伪元素子选择器出现的次数。

最后将上述 4 个数连接成一个进制很大的 4 位数 a-b-c-d，作为选择器的特指性。

在层叠处理中，如果两个声明的重要性和来源都相同，那么就要考虑它们的选择器的特指性。特指性越大（强），则优先级越高；特指性越小（弱），则优先级越低。

下面是选择器特指性的几个例子。

```
style=""                /* 特指性：1-0-0-0 */
#i2 {}                  /* 特指性：0-1-0-0 */
ul ol li.red {}         /* 特指性：0-0-1-3 */
h1 + *[type=text]{}     /* 特指性：0-0-1-1 */
li:first-line {}        /* 特指性：0-0-0-2 */
*{}                     /* 特指性：0-0-0-0 */
```

4. 声明的距离

在层叠处理中，如果两个声明的重要性、来源和其选择器的特指性都相同，那就看它定义的位置与所应用的元素的位置之间的距离，距离越近，优先级越高。这也就是所谓的就近原则。

【例 2-18】 层叠处理举例。有以下 HTML 和 CSS 代码，其中能应用于<p>元素的声明包括三个针对 color 属性的声明；两个针对 background-color 属性的声明；两个针对 font-style 属性的声明。根据层叠处理的原理，说明文档中的段落分别应用了哪个声明。

```
<head>
<style>
   .c1 {color: blue; background-color: steelblue;}
   .c2 {background-color: lightgray; font-style: italic}
   p {color: red !important; font-style: normal}
</style>
</head>
<body>
   <p class="c1 c2" style="color: green; ">北京</p>
```

```
10. </body>
```

解答：

（1）考虑声明的重要性，前景颜色设置为红色；

（2）考虑选择器的特指性，字体样式为斜体；

（3）考虑就近原则，背景颜色为浅灰色。

2.8　CSS 属性和属性值

本节介绍一些常用的 CSS 属性及其基本用法。

2.8.1　字体和文本

字体属性用于控制文本字符的显示方式，如使用的字体、大小、粗细等。这里介绍的字体属性包括 font-family、font-size、font-weight 和 font-style 等。

文本属性用于控制文本的字符间距、对齐、装饰、空白处理等。这里介绍的文本属性包括 text-align、vertical-align、letter-spacing、line-height、text-decoration 和 white-space 等。

（1）font-family。该属性用于设置字体类型。可以指定多种字体，按优先顺序排列，以逗号分隔。如果在前面指定的字体不存在相应的字体库，浏览器会考虑使用在后面指定的字体。指定的字体可以是某种具体的字体名，也可以是某种通用的字体系列名。例如：

```
font-family: 宋体;
font-family: Arial, Sans-serif;
```

该属性适用于所有元素，具有继承性。

（2）font-size。该属性用于设置字体的大小，其取值可以是长度或百分数，但不可为负值。长度是整数、浮点数和长度单位组成的值，百分数是基于父元素字体的大小来计算的。例如：

```
font-size: 14px;
```

该属性适用于所有元素，具有继承性。

（3）font-weight。该属性设置字体粗细。理论上，字体粗细分 9 级，100 为最细，900 为最粗。实际上，浏览器不一定支持这个 9 级，但会保证某级字体不会比它前一级的字体细。

也可以用标识符 normal 和 bold 作为值。默认值为 normal，相当于 400。bold 表示粗体，相当于 700。例如：

```
font-weight: bold;
```

该属性适用于所有元素，具有继承性。

（4）font-style。该属性用于设置字体风格。其取值如下。

- normal：正常字体。
- italic：斜体。
- oblique：倾斜。

例如：

```
font-style: italic;
```

该属性适用于所有元素，具有继承性。

（5）text-align。该属性设置块内文字的水平对齐方式，其取值如下。

- left：左对齐。
- right：右对齐。
- center：居中对齐。
- justify：两端对齐。

例如：

```
text-align: center;
```

该属性适用于块容器框，包括块框（除表格之外的块级框）、表格单元格、行内块，具有继承性。

（6）vertical-align。该属性设置内联元素在行框内和表格单元格内容在单元格内的垂直对齐方式。其取值可以是长度或百分数，可以是正值，也可以是负值。长度是整数、浮点数和长度单位组成的值，百分数是基于元素本身行高（line-height）来计算的。属性使元素基于基线（baseline）上升（正值）或下降（负值）指定的值。例如：

```
vertical-align: 10px;   /* 上升 10 个像素 */
```

该属性适用于内联元素、表格单元格，不具有继承性。但当应用于<tr>、<tbody>等元素时，将影响其包含的所有单元格。

（7）letter-spacing。该属性设置文本字符（包括汉字）之间的间距。属性的默认值为normal，表示由浏览器根据最佳状态设置字符间距。属性值可以取长度，即由整数、浮点数和长度单位组成的值，可以是正值，也可为负值。例如：

```
letter-spacing: 3px;
```

该属性适用于所有元素，具有继承性。

（8）line-height。对于块级元素，该属性值指定元素内各行框的最小高度，也决定了相邻行之间的间距；对于内联元素，属性值指定该元素框的高度，也是计算行框高度的基础（内联替换元素的高度应该由 height 属性指定）。属性的默认值为 normal，表示由浏览器根据元素的字体大小设置一个合理值。属性值可以取长度和百分数。百分数是基于该元素本身字体的大小来计算的。属性值也可以取数值，此时行高为该数乘以元素字体的大小。例如：

```
line-height: 20px;  /* 行高为 20 个像素 */
line-height: 1.2;   /* 行高为字体大小的 1.2 倍 */
```

该属性适用于所有元素，具有继承性。

（9）text-decoration。该属性控制是否为元素的内容文本添加装饰（用元素的前景颜色）。

其取值包括：

- underline：下画线。
- overline：上画线。
- line-through：贯穿线。
- blink：闪烁。

这些属性值可以同时设置，各属性值用空格分隔。例如：

```
text-decoration: line-through;
text-decoration: underline overline;    /* 既有下画线，又有上画线 */
```

该属性适用于所有元素，不具有继承性。通常，一个元素的该属性值会应用或传播至所有在正常流中的后代元素（内联元素或块级元素），但不包括浮动元素、绝对定位元素、行内块。

（10）white-space。该属性控制浏览器对元素内容中空白符号（空格、tab 字符和换行符）的处理。其取值及特性如表 2-17 所示。默认值为 normal。

表 2-17　white-space 属性的取值及特性

值	换行符	空白符	自动换行
normal	忽略	合并	允许
Pre	保留	保留	不允许
nowrap	忽略	合并	不允许
pre-wrap	保留	保留	允许
pre-line	保留	合并	允许

该属性适用于所有元素，具有继承性。

2.8.2　颜色和背景

颜色和背景属性用于设置元素的前景颜色、背景颜色和背景图像等。这里介绍的颜色和背景属性包括 color、background-color、background-image 和 background-attachment 等。

（1）color。该属性设置文本显示的前景颜色。指定颜色的常用方式如下。

- 颜色名：直接使用标准颜色名或浏览器支持的颜色名。
- #RRGGBB：使用两位十六进制数表示颜色中的红、绿、蓝含量。
- rgb(rrr, ggg, bbb)：使用十进制数表示颜色中的红、绿、蓝含量， 其中 rrr、ggg 和 bbb 都是 0~255 的十进制数。
- rgb(rrr%, ggg%, bbb%)：使用百分比表示颜色中的红、绿、蓝含量。

例如：

```
color: rgb(100%, 0%, 0%);    /* 红色 */
```

该属性适用于所有元素，具有继承性。

（2）background-color。该属性设置元素的背景颜色。默认值为 transparent，表示没有背景色（透明的），可以看到下层的内容。指定背景色的方式与指定前景色（color 属性）的一

样，可以是任何合法的颜色值。例如：

```
background-color: gray;
```

该属性适用于所有元素，不具有继承性。

（3）background-image。该属性用于设置元素的背景图像。例如：

```
background-image: url(bg.gif);
```

该属性适用于所有元素，不具有继承性。

（4）background-attachment。该属性用以设置背景图像相对于可视区是固定（fixed）的还是滚动（scroll）的。默认值是 scroll，此时背景图像会随文档在可视区内滚动。例如：

```
background-attachment: fixed;
```

该属性适用于所有元素，不具有继承性。

2.8.3 尺寸、边距和边框

每个 HTML 元素呈现为一个框。首先介绍一下元素的框模型，如图 2-1 所示。

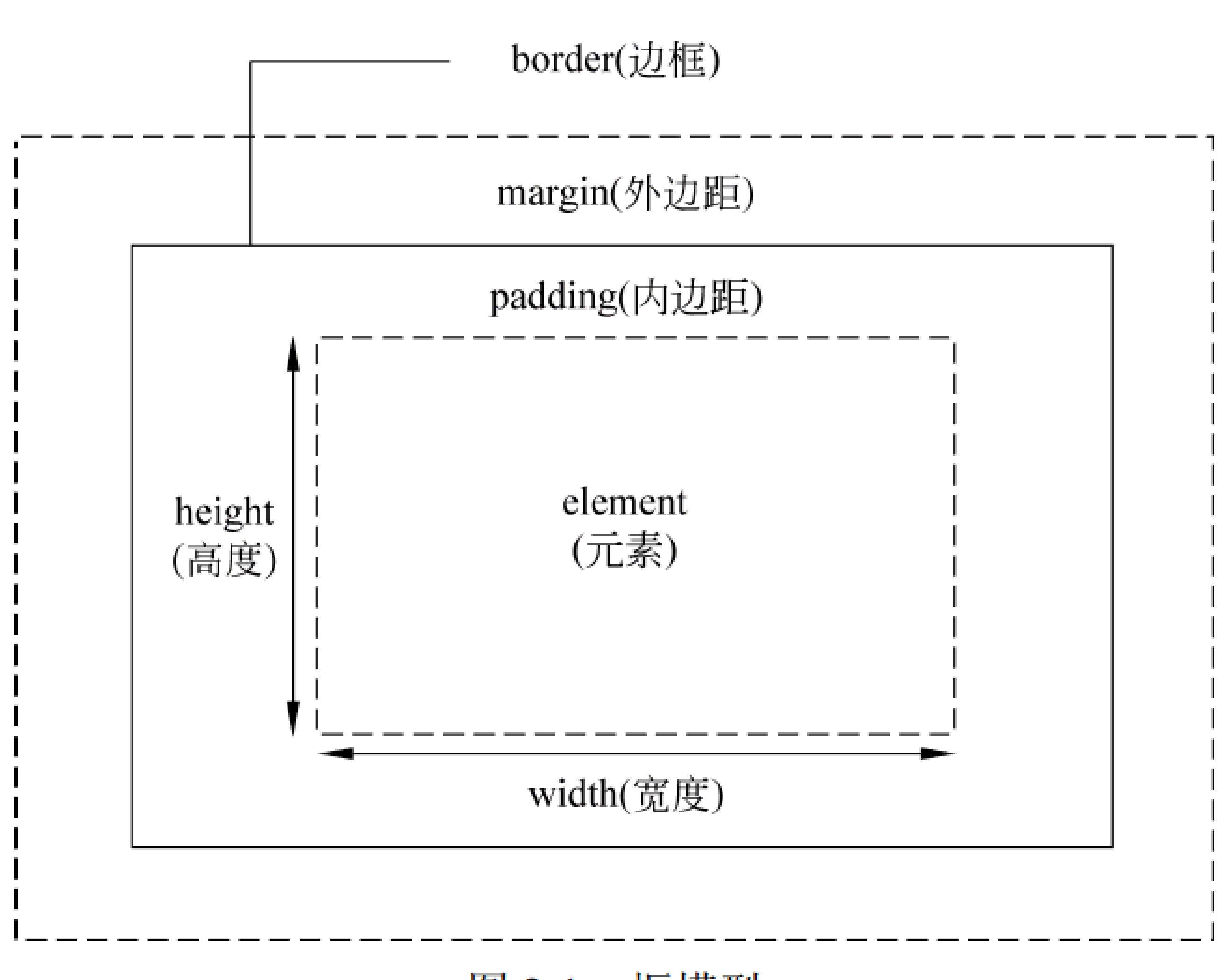


图 2-1 框模型

元素框的最内部分是实际的内容，直接包围内容的是内边距，内边距的边缘是边框，边框以外是外边距。外边距默认是透明的，因此不会遮挡其后的任何元素。

元素背景应用于由内容和内边距、边框组成的区域。边框通常有自己的颜色。

元素的 width 和 height 属性指定内容区域的宽度和高度。增加内边距、边框和外边距不会影响内容区域的尺寸，但是会增加元素框的总尺寸。

内边距、边框和外边距可以应用于一个元素的所有边，也可以应用于单独的边。

元素内容区的尺寸由 width、height、max-width、max-height、min-width 和 min-height 等属性控制。

元素内边距的尺寸由 padding 系列属性控制，外边距的尺寸由 margin 系列属性控制，

边框通过 border 属性设置。

（1）width、height。这两个属性分别设置元素框内容区的宽度和高度。默认值为 auto，表示宽度和高度由元素内容固有的宽度和高度决定。可以用长度或百分数为属性设值，不能为负值。百分数是基于包含块的宽度和高度来计算的，此时包含块应该有明确的宽度（不能是 auto）。通常，一个元素的包含块是指离该元素最近的前辈块容器框，包括块框、行内块、表格单元格。例如：

```
width: 90%;
```

width 属性不适用于非替换内联元素、表格行和行组，不具有继承性。

height 属性不适用于非替换内联元素、表格列和列组，不具有继承性。

（2）max-width、max-height。这两个属性分别设置元素框内容区的最大宽度和最大高度，默认值 none。例如：

```
max-width: 800px;
```

max-width 属性不适用于非替换内联元素、表格行和行组，不具有继承性。

max-height 属性不适用于非替换内联元素、表格列和列组，不具有继承性。

（3）min-width、min-height。这两个属性分别设置元素框内容区的最小宽度和最小高度，默认值 0。例如：

```
min-height: 500px;
```

min-width 属性不适用于非替换内联元素、表格行和行组，不具有继承性。

min-height 属性不适用于非替换内联元素、表格列和列组，不具有继承性。

（4）padding。该属性设置元素的内边距，默认值为 0。padding 属性值接受长度或百分数，但不允许使用负值。百分数是基于其包含块的 width 值来计算的。

例如：

```
padding: 10px;                /* 上、下、左、右 4 个内边距都为 10 个像素 */
padding: 10px 5px;            /* 上、下内边距为 10px，左、右内边距为 5px */
padding: 20px 5px 10px 15px;  /* 上为 20px，右为 5px，下为 10px，左为 15px */
```

该属性不适用于表行、行组、表列、列组、表头和表脚，不具有继承性。

使用 padding-top、padding-bottom、padding-left 和 padding-right 属性，可以分别设置上、下、左、右的内边距。

（5）margin。该属性设置元素的外边距，默认值为 0。margin 属性值可以是长度或百分数，甚至允许是负值。百分数是基于其包含块的 width 值来计算的。例如：

```
margin: 10px;           /* 上、下、左、右 4 个外边距均为 10px */
margin: 10px auto;      /* 上、下外边距为 10px，左、右外边距自动设置为对称 */
margin: 10px auto 0 auto;/* 上边距为 10px，右边距为 0px，左、右外边距自动设置为对称 */
```

该属性不适用于表格单元格、表行、行组、表列、列组、表头和表脚，不具有继承性。

使用 margin-top、margin-bottom、margin-left、margin-right 属性，可以分别设置上、下、

左、右的外边距。

（6）border。该属性用于为元素的 4 条边框设置相同的宽度、样式以及颜色。边框样式可以取以下值：none、hidden、dotted、dashed、solid、double、groove、ridge、inset、outset。

例如：

```
border: 5px solid red;
```

该属性适用于所有元素，不具有继承性。

使用 border-top、border-bottom、border-left 和 border-right 属性，可以分别为某条边框设置宽度、样式和颜色。

使用 border-width 属性可以为每条边框设置宽度（1~4 个值）。例如：

```
border-width: 5px;                    /* 4 条边框宽度均为 5px */
border-width: 15px 5px;               /* 上、下边框 15px，左、右边框 5px */
border-width: 15px 5px 15px 5px;      /* 上、下边框 15px，左、右边框 5px */
```

可以使用 border-top-width、border-bottom-width、border-left-width 和 border-right-width 属性，分别设置某条边框的宽度。

使用 border-style 属性可以为每条边设置样式（1~4 个值）。例如：

```
border-style: outset;                        /* 4 条边框相同的样式 */
border-style: solid dotted dashed double;    /* 每条边框具有不同的样式 */
```

可以使用 border-top-style、border-bottom-style、border-left-style 和 border-right-style 属性，分别设置某条边框的样式。

使用 border-color 属性可以为每条边框设置颜色（1~4 个值）。例如：

```
border-color: blue;
border-color: blue red;
```

可以使用 border-top-color、border-bottom-color、border-left-color 和 border-right-color 属性，分别设置某条边框的颜色。

2.8.4　定位与浮动

CSS 大致有 3 种基本的定位机制：正常流、浮动和绝对定位。除非专门指定，否则所有元素框都在正常流中定位。正常流中的元素的位置由元素在 HTML 文档中的位置决定。浮动的框可以向左或向右移动，直到它的外边缘碰到包含块或另一个浮动框的边框为止。由于浮动框不在文档的正常流中，所以正常流中的块框表现得就像浮动框不存在一样。

绝对定位的元素框相对于其包含块定位，包含块可能是文档中的另一个元素或者是初始包含块。元素原先在正常流中所占的空间会关闭，就好像该元素原来不存在一样。

除此之外，还可以将元素定位在 *z* 轴的不同位置。*z* 轴定义为垂直延伸到显示区的轴。如果为正数，则表示离用户更近；如果为负数，则表示离用户更远。

本小节介绍相关的属性包括 position、top、left、bottom、right、float、clear 和 z-index 等。

（1）position。该属性设置元素框的定位方式，下面介绍其可能的取值及相应的含义。

- static：正常定位（默认值）。在正常流中进行布局。top、right、bottom 和 left 属性不可用。
- relative：相对定位。它是指相对该元素在正常流中的位置产生一定的偏移。偏移量可由 top、left 或 right、bottom 属性指定。

相对定位框并没有脱离正常流，后续元素框在定位时也不会考虑它产生的偏移。

- absolute：绝对定位。它是指相对该元素的包含块产生一定的偏移。偏移量可由 top、left 或 right、bottom 属性指定。绝对定位元素的包含块是指离其最近的、position 属性为 absolute、relative 或者 fixed 的祖先元素，如果不存在这样的元素，则包含块为初始包含块。绝对定位框脱离了正常流，后续兄弟元素框在定位时将忽略该元素。
- fixed：固定定位。它是指相对浏览器视口（viewport）产生一定的偏移。偏移量可由 top、left 或 right、bottom 属性指定。当使用滚动条移动视口中的内容时，固定定位框不会移动。固定定位框脱离了正常流，后续兄弟元素框在定位时将忽略该元素。

position 属性适用于所有元素，不具有继承性。

（2）top、left、bottom、right。这些属性适用于相对定位、绝对定位和固定定位的元素，用于设置这些元素相对于正常位置、包含块或浏览器视口的偏移量。

（3）float。该属性指定元素框是否浮动至左侧、右侧或不浮动。其取值如下。

- left：产生一个左浮动框，向左浮动直至碰到其包含块的左边框或之前产生的另一个左浮动框的右边框。
- right：产生一个右浮动框，向右浮动，直至碰到其包含块的右边框或之前产生的另一个右浮动框的左边框。
- none：不浮动（默认值）。该属性既可应用于内联元素，也可应用于块级元素。无论是内联元素还是块级元素，浮动后都将变成浮动框，浮动框的特性类似于行内块框。该属性不具有继承性。

浮动框应尽量往高处放置，但其顶端不能高于其之前任何元素框的顶端。

（4）clear。该属性设置元素框的左右哪端不能有之前形成的浮动框。其取值如下。

- left：产生的框的顶边放置在任何之前产生的左浮动框底边框的下面。
- right：产生的框的顶边放置在任何之前产生的右浮动框底边框的下面。
- both：产生的框的顶边放置在任何之前产生的左浮动框和右浮动框底边框的下面。
- none：默认值，元素框的左右两端都允许有之前形成的浮动框。

该属性适用于块级元素，不具有继承性。

（5）z-index。该属性设置元素框在 *z* 轴上的取值（整数），值越大，越显示在前面（靠近用户），可以取正值，也可以取负值。默认情况下，初始包含块的 z-index 属性值为 0，其他元素与其父元素框取相同的值。

该属性适用于相对定位、绝对定位和固定定位的元素，不具有继承性。

【例 2-19】 创建如图 2-2 所示的水平导航栏。

该例结果包含 2 个文件：一个是页面文件，文件名为 2-19.html；一个是层叠样式表文件，文件名为 navigation.css。

下面是页面文件 2-19.html 的代码。

```
<!DOCTYPE html>
<html>
  <head>
    <title>2-19</title>
    <meta charset="UTF-8">
    <link rel='stylesheet' type='text/css' href='navigation.css'/>
  </head>
  <body>
    <div class='nav'>
      <a class='right' href='#'>退出</a>
      <span class='right'>刘绍军，您好</span>
      <a class='left current' href='#'>浏览教师信息</a>
      <a class='left' href='#'>添加课程</a>
      <a class='left' href='#'>维护开课信息</a>
      <div style='clear: both'></div>
    </div>
  </body>
</html>
```



图 2-2　水平导航栏

页面文件利用 link 标签链接外部样式表文件 navigation.css。下面是该层叠样式表文件的代码，包括“基本样式”和“水平导航栏样式”两部分。其中有些样式在本例可能没用到，但后面的实例或实验还是会用到的。

```
/* 基本样式 */
body {font-size: 14px; letter-spacing: 1.2px}
.title {color: #458994; font-weight: 700}
a {color: steelblue; text-decoration: none}
a:visited {color: steelblue}
a:hover {text-decoration: underline; font-weight: 700}
/* 水平导航栏样式 */
.nav {width:90%; background-color: #eeeeee; margin: 5px auto 5px auto}
.nav .left {float: left; padding: 5px 10px 5px 10px}
.nav .right {float: right; padding: 5px 10px 5px 10px}
.nav .current {background-color: steelblue; color: white}
```

在这里，每一个超链接元素都被呈现为一个浮动框。浮动框的特点是，既可以设置内间距、外间距，也可以设置元素的宽度和高度。另外，往同一方向浮动的框会一个挨着一个，不会因文档中存在着的空白符号，产生不可预料的间距。在例子中，第 1 个导航超链

接元素的 class 属性值既包含 left 样式类也包含 current 样式类，所以与其他超链接元素不同，它以钢青色背景、白色前景醒目显示。

2.8.5 其他属性

本小节介绍的 CSS 属性包括 display、visibility、overflow、opacity、cursor 等。

（1）display。默认情况下，HTML 的块级元素被呈现为块框，内联元素被呈现为内联框。利用该属性可以重新设置 HTML 元素生成的框的类型。其常见的取值如下。

- inline：呈现为行内框。
- block：呈现为块框。
- inline-block：呈现为行内块框。类似于替换元素，从外部看，它像一个内联框出现在内容流中；从内部看，它可以像一个块框进行格式化。
- none：元素（包括子元素）不被呈现，不产生相应的框。该属性适用于所有元素，不具有继承性。

（2）visibility。该属性指定元素框的可见性，其取值如下。

- visible：默认值。元素框是可见的。
- hidden：元素框是不可见的（完全透明），但仍影响布局。
- collapse：对表格行、行组、列、列组，产生折叠效果；对其他元素，与值 hidden 相同。该属性适用于所有元素，不具有继承性。

（3）overflow。该属性指定元素的内容溢出元素框时的处理方式，其取值如下。

- visible：默认值。溢出内容可见，越出元素框。
- hidden：溢出内容隐藏。
- scroll：显示滚动条，当溢出时，可利用滚动条查看。
- auto：滚动条根据需要显示。当溢出时，显示滚动条。

该属性适用于块框、行内块框和表格单元格，不具有继承性。

（4）opacity。该属性设置元素的不透明度，取值从 0.0~1.0，0 表示完全透明，1 表示完全不透明，默认值为 1。该属性适用于所有元素，不具有继承性。

（5）cursor。该属性用以设置当鼠标指向元素时指针的类型。其取值如下。

- auto：默认值。由浏览器设置指针。
- pointer：呈现为指示超链接的指针（手形）。
- crosshair：呈现为十字形。
- text：呈现为指示文本的指针。
- wait：呈现为指示程序正忙的指针（通常是一只表或沙漏）。
- help：呈现为帮助指针（通常是一个问号或一个气球）。

该属性适用于所有元素，具有继承性。

【例 2-20】 表单与布局。使用 CSS 技术，制作如图 2-3 所示的一个登录表单。

该例结果包含 2 个文件：一个是页面文件，文件名为 2-20.html；一个是层叠样式表文件，文件名为 formone.css。

下面是页面文件 2-20.html 的代码：



图 2-3　表单与布局

```
<!DOCTYPE html>
<html>
   <head>
      <title>2-20</title>
      <meta charset='UTF-8'>
      <link rel='stylesheet' type='text/css' href='formone.css'/>
   </head>
   <body>
      <form class='style1' method='POST'>
      <div class='outer'>
         <div class='title'>管理员登录</div>
         <div class='inter'>
            <p>
            <label for='i1' class='label'>用户名</label>
            <input type='text' id='i1' name='user' maxlength='4'
                     style='width: 60px' />
            </p>
            <p>
            <label for='i2' class='label'>密  码</label>
            <input type='password' id='i2' name='pw' maxlength='12'
                     style='width: 130px' />
            </p>
            <p style='text-align: center; padding-top: 10px'>
            <input type='submit' class='big' name='submit' value='确认'/>
            </p>
         </div>
      </div>
      </form>
   </body>
</html>
```

页面文件利用 link 标签链接外部样式表文件 formone.css。该层叠样式表文件包括“基

本样式”和“表单样式”两部分，其中“基本样式”部分已经在例 2-19 中给出，下面是“表单样式”部分。有些样式在本例可能没用到，但后面的实例或实验还是会用到的。

```
/* 表单样式 */
form {margin: 10px 0}
form .label {display: inline-block; width: 70px; margin-right: 10px}
form.style1 {text-align: center}
form.style1 div.outer {border: 1px solid #458994; display: inline-block}
form.style1 div.title {
   color: white; background-color: #458994; font-size: 20px;
   padding-top: 12px; padding-bottom: 8px; margin: 1px
}
form.style1 div.inter {padding: 10px; text-align: left}
input, select {font-size: 13px; padding: 3px 1px}
input[type=submit] {
   padding: 4px 15px; color: steelblue; background-color: #eeeeee;
   border: 1px solid steelblue; cursor: pointer
}
input[type=submit].big {font-size: 15px; padding: 7px 30px}
input[type=submit].text {
   font-size: 14px; padding: 1px 2px; color: steelblue;
   background-color: white; border-width: 0
}
input[type=submit]:hover {font-weight: 700}
textarea {letter-spacing: 1.2px; line-height: 1.5}
form .errMsg {font-size: 13px; color: red}
```

该例用 CSS 技术实现表单的布局，其要点如下：

（1）<form>元素是一个块级元素，其宽度是整个容器的宽度。该<form>元素采用了样式类"style1"，它使表单框内的内容水平居中显示。这里把该<form>元素形成的块框称为表单框。

（2）<form>元素内有一个<div>子元素，采用了样式类"outer"，它使该<div>元素呈现为一个行内块框，所以其宽度由其内容的实际宽度决定。这里把该<div>元素形成的框称为外框，外框呈现时有边框线，在表单框内水平居中显示。

（3）外框内包含两个<div>元素。第 1 个<div>元素框用于显示表单标题。标题在框内水平居中显示，该特性从<form>元素继承而来。

（4）外框内的第 2 个<div>元素采用了样式类"inter"，它使框中的内容左对齐。这里把该<div>元素形成的块框称为内框。内框包含所有表单控件元素，每个控件元素放置一个段落元素中。由于 CSS 属性的继承性，段落内的内容也是左对齐。

（5）所有<label>元素被呈现为一个行内块框，这样就可以为其设置宽度。所有标签的宽度、边距都是相等的，这样在标签右侧的文本域等表单控件元素就可以做到左对齐。由于 CSS 属性的继承性，所有标签文字在标签宽度内左对齐。

习 题 2

1. 根据要求写 CSS 规则

（1）id 属性值为"id1"的元素显示边框线：宽度为 1px 的蓝色实线。

（2）<div>元素内的所有段落子元素的文字：字符间距 2px，行高为字符大小的 1.2 倍。

（3）<body>元素内的所有<div>子元素的宽度为浏览器窗口宽度的 90%，居中显示。

（4）class 属性值既包含类名 c1 又包含类名 c2 的元素呈现为左浮动框，内边距为 5px。

（5）当鼠标指向 class 属性值为 c1 的段落元素时，指针显示为手形。

2. 简答题

（1）根据定义的 CSS 规则，说明下面 HTML 文档中文字的呈现格式。

```
<head>
  <style type='text/css'>
    span {color: green}
    .c {border: 1px solid blue}
  </style>
</head>
<body>
  <span class='a b c'>Web 应用开发</span>
</body>
```

（2）根据定义的 CSS 规则，说明下面 HTML 文档中各段落的呈现格式。

```
<head>
  <style type='text/css'>
    p.c {color: red; border: 1px solid blue}
    p {color: yellow}
  </style>
</head>
<body>
  <p>段落一</p>
  <p class='c'>段落二</p>
  <p class='c' style='color: green'>段落三</p>
</body>
```

（3）根据定义的 CSS 规则，说明下面 HTML 文档中各段落文字的呈现颜色。

```
<head>
  <style type='text/css'>
    div>p.c {color: yellow}
    div p {color: red}
    div+p {color: green}
  </style>
</head>
```

```
<body>
  <div>
    <p>段落一</p>
    <div><p class='c'>段落二</p></div>
  </div>
  <p>段落三</p>
</body>
```

（4）在下面 HTML 元素中，哪些是替换元素？

<a>、<img>、<span>、<div>、<label>、<input>、<textarea>、<select>、
、<p>

（5）在 CSS 层叠处理中，“就近原则”指什么？什么情况下应用“就近原则”？

第 3 章　数据与变量

本章主题：

- 数据类型；
- 文字的表示；
- 类型的自动转换与强制转换；
- 变量与变量赋值；
- 变量作用域；
- 可变变量；
- PHP 常量。

数据是计算机处理的对象，每一个数据都属于某种类型。类型规定了数据的性质、取值范围以及在其上可以进行的操作行为。一种计算机程序设计语言能够支持的数据类型的多少，很大程度上决定了这种语言的功能强弱。

对于任何一种计算机程序设计语言，变量都是非常基本且又重要的概念。学会使用变量是计算机程序设计的基础。

本章首先介绍 PHP 的各种数据类型以及类型之间的转换，然后介绍变量、变量作用域以及常量等的概念和使用。

3.1　PHP 数据类型

在 PHP 中，数据类型分为标量类型、复合类型和 NULL 类型。

3.1.1　标量类型

标量类型数据表示某种单项信息，包括布尔型、整型、浮点型和字符串型。

1. 布尔型

布尔型（Boolean 或 Bool）是一种最简单的数据类型，其数据值只有两个：true 和 false，分别表示逻辑真和逻辑假。这里，true 和 false 是不区分大小写的。

在很多需要布尔型数据的场合，其他类型的数据会被自动转换成布尔值，如非零值被转换成 true，零值（0、0.0）被转换成 false。

典型地，布尔型数据经常被用在流程控制语句中。

【例 3-1】 布尔型数据的使用。代码如下：

```
<?php
$alive = true;
if($alive) {     // 条件为真
  echo "This cat is alive.";
```

```
}
$alive = 0;
if($alive) {    // $alive 自动转换成布尔型数据 false
    echo "The dog is alive.";
} else {
    echo "The dog is dead.";
}
?>
```

下面是代码运行的输出结果：

```
This cat is alive.The dog is dead.
```

该例中，变量$alive 用作 if 语句的条件表达式。在第 2 条 if 语句中，变量$alive 的值为整数 0，会自动转换成布尔型数据 false。

用 print 或 echo 输出布尔型数据时，true 被转换成字符串"1"，false 被转换成空串。为了显示出包括布尔型数据在内的各种类型数据本来的内容，可以使用 var_dump 函数。

```
void var_dump(mixed $expression1 [, mixed $expression2]*)
```

该函数用于输出指定的一个或多个表达式的结构化信息，包括表达式的类型与值。

【例 3-2】 布尔型数据的输出。代码如下：

```
<?php
$var1 = true;
$var2 = false;
echo "-", $var1, $var2, "-";
echo "<pre>";
var_dump($var1, $var2);
echo "</pre>";
?>
```

下面是代码的运行结果：

```
-1-
bool(true)
bool(false)
```

var_dump 函数主要用于程序调试，它会按预定义的格式输出各表达式的类型及值。有时候这种格式会包含一些普通的换行符（\r、\n 等），它们无法在浏览器上反映出来，此时可以将 var_dump 函数放置在 echo "<pre>"和 echo "</pre>"之间，如该例代码所示。通常，人们只需关注各表达式的相关信息，无须在意这些信息的呈现格式。

2. 整型

整型（integer、int）数是不包含小数部分的数。整型文字最前面可以有一个“+”或“-”，之后由一些数字组成。在 PHP 中，整数可用十进制、八进制（以 0 开头）、十六进制（以 0x 开头）或二进制（以 0b 开头）表示。例如：

```
$n1 = 0;          // 零
$n2 = -28;        // 负整数
$n3 = 056;        // 八进制数（等于十进制数 46）
$n4 = 0xA;        // 十六进制数（等于十进制数 10）
$n5 = -0x1c;      // 十六进制数（等于十进制数-28）
$n6 = 0b1001;     // 二进制数（等于十进制数 9）
```

在十六进制表示中，前导符 x 大小写均可，表示数字的符号 a~f 也是大小写均可。在二进制表示中，前导符 b 也是大小写均可。

PHP 支持的最大整数与平台有关。预定义常量 PHP_INT_SIZE 能够返回整型数字长的字节数；预定义常量 PHP_INT_MAX 返回最大整型数。自 PHP 7.0.0 版本后，可以使用预定义常量 PHP_INT_MIN 返回最小整型数。如对于 32 位字长的整数，常量 PHP_INT_SIZE 返回 4，常量 PHP_INT_MAX 返回 2147483647，常量 PHP_INT_MIN 返回 –2147483648。如果一个整数超出了整型的值域范围，将会自动转换成浮点型（float）。

【例 3-3】 整数溢出。代码如下：

```
<?php
echo PHP_INT_SIZE, "<br />";
echo PHP_INT_MAX, "<br />";
echo "<pre>";
var_dump(0x7fffffff);
var_dump(0x80000000);
var_dump(0xffffffff);
var_dump(100*21474837);
echo "</pre>";
?>
```

当整数字长是 32 位的情况下，上面代码的运行输出结果如下：

```
4
2147483647
int(2147483647)
float(2147483648)
float(4294967295)
float(2147483700)
```

3. 浮点型

浮点型（float、double 或 real）数是可以表示包含小数的数。浮点型数只有十进制表示形式，但有标准计数法和科学计数法两种表示方法。例如：

```
$a = 1.234;
$b = -38.;
$c = .25;
$d = -1.2e3;
$e = 7E-10;
```

其中，标准计数法由整数部分和小数部分组成，两者可省略其一，但应该包含小数点。在科学计数法中，E 或 e 前面必须有数字（可以是整数），E 或 e 后面必须是整数（正号可省略）。

PHP 支持的最大浮点数与平台有关，通常是 1.8e308，且具有约 14 位十进制数字的精度。如果一个浮点数的绝对值太大而无法表示，其结果将为正无穷或负无穷。可以用函数 is_infinite 判断一个浮点数是否为无穷大（正或负）。有些浮点数运算会产生一个特殊 NaN 值，该值表示一个在浮点数运算中未定义的值，如一个正无穷减去一个正无穷。可以用函数 is_nan 判断一个浮点数是否为 NaN 值。

浮点数及其运算存在着精度问题，即可能会出现一定的误差，所以一般不要比较两个浮点数是否严格相等。通常可以在一定的误差范围内比较两个浮点数是否相等。

【例 3-4】 浮点数比较。代码如下：

```
<?php
$a = 1.23456789;
$b = 1.23456780;
$epsilon = 0.0000001;
if(abs($a-$b) < $epsilon) {
   echo "true";
} else {
   echo "false";
}
?>
```

下面是代码运行的输出结果：

```
true
```

代码中 abs 是一个求绝对值的 PHP 内置函数。

4. 字符串型

字符串型值（即字符串）是指一串字符。在写字符串文字时，应该用单引号或双引号作为定界符，将这串字符括起来。例如：

```
$s1 = "example1";
$s2 = 'example2';
```

在写字符串时，通常可以用转义字符（以反斜杠\开头）表示一些特殊的字符，比如用\n 表示换行符，用\r 表示回车符等。

需要注意的是，双引号字符串和单引号字符串在处理转义字符上是有区别的。双引号字符串可以使用的转义字符如表 3-1 所示。

表 3-1 双引号字符串可以使用的转义字符

转义字符	说　　明
\n	换行符（0x0A）
\r	回车符（0x0D）

续表

转义字符	说　明
\t	水平制表符（0x09）
\v	垂直制表符（0x0B）
\e	ESC 字符（0x1B）
\f	换页符（0x0C）
\\	反斜杠
\$	美元符号
\"	双引号
\[0-7]{1,3}	1~3 位八进制数表示的一个字符
\x[0-9A-Fa-f]{1,2}	1~2 位十六进制数表示的一个字符

在双引号字符串中可以直接使用单引号，不要使用转义字符（\'），否则会被处理成两个字符，即反斜杠（\）和单引号（'）。

单引号字符串仅可使用的转义字符有两个：“\\”和“\'”，分别表示反斜杠和单引号。其他转义字符不会被特殊处理。

另外，当双引号字符串中出现变量时，PHP 会自动将其替换成变量的值并转换成字符串，而单引号字符串则无此特性。

【例 3-5】 转义字符和变量在字符串中的使用。代码如下：

```
<?php
$output1 = "This is one line.\nThis's another line.\n";
echo $output1;
$output2 = 'This is one line.This\'s another line.\n';
echo $output2, "<br />";
$fruit = "apple";
echo "\xaThis is an $fruit.\n";
echo 'This is an $fruit.';
?>
```

上述代码运行的输出结果在浏览器窗口中的显示效果如下：

```
This is one line. This's another line. This is one line.This's another line.\n
This is an apple. This is an $fruit.
```

在浏览器源代码窗口中的显示效果如下：

```
This is one line.
This's another line.
This is one line.This's another line.\n<br />
This is an apple.
This is an $fruit.
```

一般情况下，在浏览器窗口中，一个或多个空格和新行符仅显示为一个空格。要在新

的一行显示内容，可使用 HTML 标签
。在源代码窗口中，碰到换行符，会另起一行显示，而
则被当作一般的字符显示。

如果要在双引号字符串中输出变量的值，而变量名与相邻内容无法分隔时，可以用花括号界定变量名。例如：

```
$x = "abc";
echo "123$xyz";      // 变量$xyz 找不到，输出：123
echo "123{$x}yz";    // 输出：123abcyz
```

有关字符串的各种处理函数和方法以及正则表达式的内容将在第 6 章作详细介绍。

3.1.2　复合类型

复合类型数据通常是多项信息的集合，每项信息有各自的数据类型。在 PHP 中，复合类型包括数组和对象两种。

1. 数组

在 PHP 中，数组是有序的映射。一个数组由若干元素组成，每个元素是一个键-值对。其中，键用于索引数组元素，一个数组不能有重复的键。键可以是整数也可以是字符串。值也称为数组元素的值，可以是任意类型。

与标量类型一样，数组类型并不需要定义。需要时，可以直接创建数组。例如：

```
$arr1 = array(0 => "浙江", 1 => "江苏", 2 => "广东");
$arr2 = array("浙江" => "杭州", "江苏" => "南京", "广东" => "广州");
```

上面代码通过 array()依次创建两个数组并分别赋给了变量$arr1 和$arr2。数组$arr1 有 3 个键-值对，其中键是整数。数组$arr2 也有 3 个键-值对，其中键是字符串。通过一个具体的键可以访问对应的值，比如$arr1[0]将返回"浙江"，$arr2["浙江"]将返回"杭州"。

有时候，把键为整数的数组称为数字索引数组，把键为字符串的数组称为关联数组。

有关数组的创建、访问、遍历、排序和使用等具体内容会在第 10 章进行详细介绍。

2. 对象

确切地说，对象并不是一种数据类型，而是某种类类型的实例。也就是说，类是一种数据类型，而对象是这种类型的值。

与其他数据类型不同，类需要显式地进行定义。类是对一类相似对象的描述，这些对象具有相同的属性和行为、相同的变量（数据结构）和方法实现。类定义就是对这些变量和方法实现进行描述。类好比是一类对象的模板，有了类定义后，基于它就可以生成这种类型的任何一个对象。这些对象虽然采用相同名字的变量来表示状态，但它们占有各自的内存空间，在变量上的取值完全可以不同。这些对象一般有着不同的状态，且彼此间相对独立。

与其他类型的值不同，对象既有表示状态的变量值，也有表示行为的方法代码。外界通常通过访问其方法代码与对象进行交互。调用一个对象的方法意味着该对象的某种行为的发生。执行对象方法时往往需要访问该对象的状态数据，即变量的值。有时候仅仅是读取变量的值，有时候则可能改变变量的值。

【例 3-6】 类定义和对象创建。代码如下：

```
<?php
class Circle {
private $radius;
function setRadius($r) {
   $this->radius = $r;
}
function getPerimeter() {
   return 2*3.14159*$this->radius;
}
}
$c = new Circle();
$c->setRadius(1);
$p = $c->getPerimeter();
echo "The perimeter is $p.";
?>
```

下面是代码运行的输出结果：

```
The perimeter is 6.28318.
```

使用关键字 class 来定义类，关键字后面指定类名，然后是一对花括号。花括号内是类体，定义对象的状态变量和方法代码。

上面代码首先定义了一个表示圆的名为 Circle 的类。在类体中，定义了一个表示圆半径的变量$radius，一个用于设置圆半径的方法 setRadius()，以及一个能够计算并返回圆周长的方法 getPerimeter()。

后半段代码先用 new 运算符创建了一个圆对象，然后设置该圆的半径，最后计算并输出圆的周长。

有关类和对象的具体内容将在第 11 章进行详细介绍。

3.1.3 NULL 类型

在 PHP 中，NULL 类型是一种特殊类型。NULL 类型的唯一值是 NULL，代表无值。NULL 是不区分大小写的。

一个变量在下面情况下具有 NULL 值：

（1）被赋予常量 NULL。

（2）不存在。

（3）被 unset 销毁的变量。

其中，后面两种情况实际上是一回事，即变量不存在，或者说变量没有定义。当访问一个没有定义的变量时，虽然认定其值为 NULL，但会产生一个 Notice 错误信息。这里，unset 是一种语言结构，其语法格式如下：

```
void unset(mixed $var1 [, mixed $var2]*)
```

unset 用以销毁指定的一个或多个变量。

可以使用 is_null 函数来测试一个变量的值是否为 NULL，其语法格式如下：

```
bool is_null(mixed $var)
```

如果$var 没有定义或其值是 NULL，函数返回 true，否则返回 false。如果被检测的变量没有定义，is_null 函数会产生一个 Notice 错误信息。

另外有两个语言结构可以测试变量是否设置或是否为空：isset 和 empty。其中，isset 语言结构用以测试变量是否设置，其语法格式如下：

```
bool isset(mixed $var1 [, mixed $var2]*)
```

检测指定变量是否设置。所谓一个变量已设置是指该变量已经定义并且值不是 NULL。若变量已设置，语言结构返回 true，否则返回 false。如果指定多个变量，则仅当所有变量都已设置，语言结构返回 true，否则返回 false。

另一个语言结构是 empty，其语法格式如下：

```
bool empty(mixed $var)
```

用以判断指定变量是否为空。如果指定变量没有定义或者它的值为空，则语言结构返回 true，否则返回 false。

一个变量的值为空是指该变量具有以下值的情况：

- ""（空字符串）。
- "0"（作为字符串的 0）。
- 0（作为整数的 0）。
- 0.0（作为浮点数的 0）。
- NULL。
- false。
- array()（一个空数组）。

【例 3-7】 NULL、空及相关函数。代码如下：

```
<?php
$a = 0;
$b = null;
var_dump(is_null($a), is_null($b), is_null($c));   //false, true, true
var_dump(isset($a), isset($b), isset($c));         //true,false,false
var_dump(empty($a), empty($b), empty($c));         //true,true,true
?>
```

这里，由于变量$c 没有定义，所以在计算 is_null($c)时会产生一个 Notice 错误信息。

说明：

（1）就检测一个变量来说，is_null 与 isset 的检测结果正好相反。但 isset 可以检测多个变量。

（2）empty 检测变量的值是否为空。这里“空”包含 NULL 但范围更广，还包含各种

类型的一些“特殊”数据，如数值 0 等。

（3）当检测一个未定义变量时，is_null 会产生一个 Notice 错误信息，而 isset 和 empty 都不会给出错误信息。

3.2 类 型 转 换

类型转换是指将某种类型的数据转换成另一种类型的数据。类型转换包括自动类型转换和强制类型转换。

3.2.1 自动类型转换

PHP 是一种弱类型的编程语言，在引入变量时，并不需要明确声明其类型。当给它赋一个字符串时，变量的类型就是字符串型；如果再给它赋一个整数，那么变量的类型就变为整型的。

有时候在计算表达式时，PHP 会根据上下文（如运算符，函数或流程控制语句）自动将一些数据转换成合适的类型来参与运算。例如，在做加法（+）运算时，如果某个操作数不是数值型（浮点型或整型），那么 PHP 会自动将其转换成数值型，然后再进行运算，运算结果是数值型的。

1. 自动转换为布尔型

在需要布尔型数据的场合，其他类型的数据通常会被自动转换成布尔型。下面数据将被转换成布尔值 false：

- 整型值 0（零）。
- 浮点型值 0.0（零）。
- 空字符串，以及字符串"0"。
- 不包括任何元素的数组。
- NULL 值（包括没有定义的变量）。

上述相应类型的其他值则被转换成布尔值 true。对象总是被转化成布尔值 true。

2. 自动转换成数值型

在做算术运算或其他需要数值型数据的场合，其他类型的数据会自动转换成整数或浮点数：

- 布尔值 true 转换为 1，布尔值 false 转换为 0。
- 字符串可以根据其开始部分转换成整数、浮点数或零。

3. 自动转换成字符串

在做字符串连接运算或其他需要字符串的场合，其他类型的数据会被自动转换成字符串：

- 布尔值 true 转换成字符串"1"，布尔值 false 转换成空串" "。
- 整数或浮点数转换成该数值字面样式的字符串。
- 数组转换成字符串"Array"。
- 在 PHP5 及之后，对象被转换成该对象的__toString 方法的返回值。

- NULL 值转换成空串" "。

【例 3-8】 自动类型转换。代码如下：

```
1.  <?php
2.  $a = "abc";
3.  $b = "12.5xyz";
4.  $c = true;
5.  if($a) {
6.     $d = 10+$b;
7.  } else {
8.     $d = 10+$c;
9.  }
10. $result = $a."-".$b."-".$c."-".$d;
11. echo $result;
12. ?>
```

下面是代码运行的输出结果：

```
abc-12.5xyz-1-22.5
```

在该示例代码中，作为if语句的条件表达式，变量$a的值"abc"被自动转换成布尔值true。在做算术运算时，变量$b 的值"12.5xyz"将被转换成浮点数 12.5；变量$c 的值 true 会被转换成整数 1。由于第 5 行的 if 条件成立，所以程序运行时第 6 行代码被执行，第 8 行代码没有被执行。

第 10 行代码中的点（.）是字符串连接运算符。在做字符串连接运算时，变量$c 的值 true 将被转换成字符串"1"；变量$d 的值 22.5 会被转换成字符串"22.5"。

3.2.2 强制类型转换

如上所述，一些数据可以从一种类型自动转换成另一种类型。有些时候，也可以进行强制类型转换。强制类型转换的语法是：在要转换的变量（或表达式）之前加上用圆括号括起来的目标类型。

1. 转换成布尔型

要把一个表达式的值强制转换成布尔型，语法格式如下：

```
(bool)<表达式>
```

或

```
(boolean)<表达式>
```

强制转换成布尔型的转换规则与自动转换成布尔型的是相同的。

2. 转换成整型

强制转换成整型的语法格式如下：

```
(int)<表达式>
```

或

```
(integer)<表达式>
```

强制转换成整型的转换规则：

（1）布尔值 true 转换为 1，布尔值 false 转换为 0。

（2）字符串根据其开始部分转换成整数或 0。

（3）浮点数转换为整数时，抛弃小数位取整。如果浮点数超出了整型的值域范围，则结果是不确定的。

从其他类型转换成整数，PHP 没有明确定义。

3. 转换成浮点型

强制转换成浮点型的语法格式如下：

```
(float)<表达式>
```

或

```
(double)<表达式>、(real)<表达式>
```

强制转换成浮点型的转换规则：

（1）字符串根据其开始部分转换成浮点数或 0。

（2）对其他类型的值，先将其转换成整数，然后再转换成浮点数。

4. 转换成字符串

强制转换成字符串型的语法格式如下：

```
(string)<表达式>
```

强制转换成字符串型的转换规则与自动转换成字符串型的是相同的。

5. 转换成数组

强制转换成数组的语法格式如下：

```
(array)<表达式>
```

强制转换成数组的转换规则：

（1）一个任意标量类型（integer、float、string 或 boolean）的值转换成数组，将得到一个仅有一个元素的数组：该元素的下标为 0，其值为此标量类型的值。

（2）一个对象转换为数组，将得到如下一个数组：对象的各实例变量转换为数组元素，其中实例变量名为元素键，实例变量值为元素值。如果实例变量是非公共的，可能会导致一些不可预知的行为。

（3）将 NULL 值转换为数组，将得到一个不含元素的空数组。

6. 转换成对象

强制转换成对象的语法格式如下：

```
(object)<表达式>
```

将一个对象转换成对象，不会有任何变化。将任何非对象类型的值转换成对象，将会创建一个内置类 stdClass 的实例：

（1）如果该值为 NULL，则新的实例为空。

（2）数组转换成对象时，每个数组元素对应一个实例变量，其中元素键作为变量名，元素值作为变量值。

（3）对于任何其他类型的值，名为 scalar 的实例变量将包含该值。

7. 转换成 NULL

下面格式可以将任何类型的数据转换成 NULL 类型的值，即 NULL 值：

```
(unset)<表达式>
```

【例 3-9】 强制类型转换。代码如下：

```
<?php
$a = "abc";
$b = "12.5xyz";
$c = true;
if($a) {
   $d = 10 + (integer)$b;
} else {
   $d = 10 + $b;
}
if((unset)$a) {
   $e = 10 + $c;
} else {
   $e =10 + (float)$c;
}
var_dump($d, $e);
?>
```

下面是代码运行的输出结果：

```
int(22)
float(11)
```

在该例代码中，(integer)$b 会将变量$b 的值"12.5xyz"转换成整数 12；(unset)$a 会将变量$a 的值转换成 NULL 值，然后再自动转换成布尔值 false；(float)$c 会先将变量$c 的值 true 转换成整数 1，再转换成浮点数 1.0。

3.3 变量与常量

在任何计算机程序设计中，学会使用变量都是必不可少。本节首先介绍变量的概念，然后介绍变量的赋值、变量的作用域及可变变量等知识，最后介绍常量的定义与使用。

3.3.1 PHP 变量

变量是具有名字的内存单元，其中存有数据，称为变量值。一个变量在不同时刻可以存储不同的值，通过变量名可以访问变量的当前值。

PHP 中的变量用一个美元符号后面跟变量名来表示。变量名遵循标识符的命名规则，即以字母或下画线开头，后跟任意数量的字母、数字和下画线。变量名是大小写敏感的。

PHP 是一种弱类型语言，或者称为动态类型语言。在 PHP 中，变量不需要显式声明，也没有固定的类型。变量的类型由当前赋给变量的值确定。例如，下面代码可以创建变量$totalamount，其初始类型为浮点型：

```
$totalamount = 0.00;
```

如果接着执行下面代码：

```
$totalamount = "Hello";
```

那么，变量$totalamount 的类型就变成是字符串型的。PHP 可以在任何时刻根据保存在变量中的值来确定变量的类型。

3.3.2 变量赋值

变量赋值有两种方式：值赋值和引用赋值。

默认情况下，变量按值赋值，即将赋值运算符右侧的表达式的值赋给赋值运算符左侧的变量。这意味着，把一个变量的值赋给另一个变量后，如果改变其中一个变量的值，并不会影响另一个变量的取值。

引用赋值要求赋值运算符左右两侧都是变量，左侧的称为目标变量，右侧的称为源变量。引用赋值是指将源变量的引用赋给目标变量，其结果是目标变量与源变量引用相同的内存单元。之后，改变其中一个变量的值，将反映到另一个变量上。在源变量名前加上一个“&”符号将实现按引用赋值。

【例 3-10】 值赋值和引用赋值。代码如下：

```
<?php
$number1 = 22;
$age1 = $number1;
$number1 = 30;
echo "\$number1=$number1, \$age1=$age1";
echo "<br />";
$number2 = 22;
$age2 = &$number2;
$number2 = 30;
echo "\$number2=$number2, \$age2=$age2";
?>
```

下面是代码运行的输出结果：

```
$number1=30, $age1=22
$number2=30, $age2=30
```

3.3.3 变量作用域

变量作用域是指变量可被访问的范围。超出了变量的作用域，该变量即不复存在。按作用域分，PHP 变量包括：全局变量、局部变量、静态变量和超全局变量。

1. 全局变量

在用户自定义函数外的脚本中引入的变量是全局变量，其作用域是它所在的整个文件，但不包括其中的用户自定义函数内部。

2. 局部变量

在用户自定义函数内引入的变量是局部变量，其作用域是该函数内部。当退出函数时，这些变量将被清除。

【例 3-11】 全局变量与局部变量。代码如下：

```
<?php
$a = 100;
echo $a."<br/>";
test();
echo $a."<br/>";

function test() {
   $a = 'abc';
   echo $a."<br/>";
}
?>
```

下面是代码运行的输出结果：

```
100
abc
100
```

在该例代码中，函数外面声明的$a 是一个全局变量，在函数外的所有代码中是有效的，但在函数内无效。函数内的$a 是一个新的局部变量，与函数外的变量$a 无关，只在函数内有效。

也可以在函数内使用关键字 global 声明新的全局变量或使用已有的全局变量。例如：

```
global $x;
```

这样，如果在调用函数前已经存在同名的全局变量$x，那么在函数内访问的$x 就是那个已经存在的同名变量。如果在调用函数前并不存在同名的全局变量$x，那么该变量将在函数内创建，且在退出函数后仍然有效。

【例 3-12】 使用 global 关键字。代码如下：

```php
<?php
$a = 100;
echo $a."<br/>";
test();
echo $a."<br/>";
echo $b."<br/>";

function test() {
   global $a,$b;
   $a = 'abc';
   $b = 'hello';
}
?>
```

下面是代码运行的输出结果：

```
100
abc
hello
```

说明：在函数外的脚本代码中，也可以用 global 显式声明一个变量是全局的。在被包含的文件中这样声明是有用的。有关 PHP 文件的包含与被包含的技术会在 4.4 节介绍。

3. 静态变量

在函数内，可以用 static 声明并初始化一个变量，称为静态变量。

```php
static $a = 100;
```

与局部变量相同，静态变量也仅在函数内有效，但有以下不同：

（1）静态变量的初始化仅发生在第 1 次调用函数时；

（2）退出函数后，静态变量不会被清除，其值被保存。

【例 3-13】 使用静态变量。代码如下：

```php
<?php
test();
test();

function test() {
   static $a = 100;
   $a++;
   echo $a."<br/>";
}
?>
```

下面是代码运行的输出结果：

```
101
102
```

第 1 次调用 test()函数时，静态变量$a 被创建并初始化为 100，增 1 后变为 101。第 2 次调用该函数时，变量$a 已经存在且不会初始化，增 1 后由原来的 101 变为 102。

4. 超全局变量

超全局变量是指一些系统预定义变量，它们在所有的 PHP 文件的脚本代码中都是有效的，在这些脚本代码中的自定义函数外或自定义函数内都是可用的。超全局变量包括以下几种。

（1）$GLOBALS：包含当前所有全局变量（包括其他超全局变量）的数组。

（2）$_SERVER：包含服务器及执行环境信息的数组。

（3）$_GET：包含通过 GET 方法传递给当前脚本代码的请求参数的数组。

（4）$_POST：包含通过 POST 方法传递给当前脚本代码的请求参数的数组。

（5）$_COOKIE：包含传递给当前脚本代码的 Cookies 的数组。

（6）$_FILES：包含传递给当前脚本代码的上传文件相关信息的数组。

（7）$_SESSION：包含会话变量的数组。

这里，介绍一下$GLOBALS 和$_SERVER 两个超全局变量的使用。其他几个超全局变量在后面的相关章节会详细介绍。

利用超全局变量$GLOBALS，可以访问全局变量。下面代码访问一个名为$var 的全局变量：

```
$GLOBALS['var'] = 10;
```

该代码既可以放置在函数外，也可以放置在函数内。如果之前存在全局变量$var，那么代码会给该变量赋整数 10。如果之前并不存在全局变量$var，那么代码将创建这样一个全局变量并给它赋值。

【例 3-14】 使用超全局变量$GLOBALS。代码如下：

```
<?php
function test() {
   if(isset($GLOBALS['var'])) {
      $GLOBALS['var'] += 10;
   } else {
      $GLOBALS['var'] = 100;
   }
}

test();
var_dump($var);
test();
var_dump($var);
?>
```

下面是代码运行的输出结果：

```
int(100)
```

```
int(110)
```

超全局变量$_SERVER 是一个包含服务器及执行环境信息的数组。通过它可以访问诸如请求头信息(Header)、路径(Path)、脚本位置(Script Locations)等信息。比如要了解当前请求的方法，可以访问数组元素$_SERVER['REQUEST_METHOD']获得。下面列出了一些元素的键，通过访问这些键的特定元素，可以获取相关的信息。

- 'DOCUMENT_ROOT'：Web 服务器设置的文档根目录，例如 C:/xampp/htdocs。
- 'SCRIPT_NAME'：当前被请求的 PHP 文件相对于文档根目录的路径和文件名。
- 'SCRIPT_FILENAME'：当前被请求的 PHP 文件的绝对路径和文件名。
- 'REQUEST_METHOD'：当前请求采用的请求方法，例如 GET、POST。
- 'SERVER_PROTOCOL'：当前请求采用的通信协议的名称和版本，例如 HTTP/1.0。
- 'REQUEST_URI'：当前请求行中 URI 的内容。
- 'QUERY_STRING'：查询串，即 GET 请求的请求参数，包含在 URI 的尾部。
- 'HTTP_HOST'：当前请求头中 Host 域的值，例如 localhost:8080。
- 'HTTP_ACCEPT'：当前请求头中 Accept 域的值，表示用户代理可接受的 MIME 类型。
- 'HTTP_ACCEPT_CHARSET'：当前请求头中 Accept-Charset 域的值，表示用户代理可接受的字符集。
- 'HTTP_ACCEPT_LANGUAGE'：当前请求头中 Accept-Language 域的值，表示用户代理可接受的语言。
- 'HTTP_REFERER'：当前请求头中 Referer 域的值，表示引导用户代理到当前页的前一页的地址（如果存在）。

说明：这些键是否有效，或者说$_SERVER 中是否存在相应的元素，取决于 Web 服务器是否对它进行了创建和提供，另外也取决于请求头中是否包含相关的信息。

3.3.4 可变变量

可变变量是指变量名可变的变量，或者说一个可变变量在不同时刻可能代表不同的变量。可变变量由在变量名前加两个美元符（$）来表示，一个可变变量获取一个普通变量的值作为这个可变变量的变量名。

【例 3-15】 使用可变变量。代码如下：

```
<?php
$a = "zj";
$$a = "杭州";
echo $a." ".$$a." ".$zj."<br/>";

$a = "js";
$$a = "南京";
echo $a." ".$$a." ".$js."<br/>";
?>
```

下面是代码运行的输出结果：

```
zj 杭州 杭州
js 南京 南京
```

在该例代码中，$$a 是一个可变变量。当第 1 次给它赋值时，它代表变量$zj；当第 2 次给它赋值时，它代表变量$js。

当将可变变量用于数组时，需要解决一个二义性问题。例如$$a[1]，是把数组元素$a[1]作为可变变量的名，然后访问该可变变量；还是把$a 作为可变变量的名，然后访问该可变变量中索引为 1 的元素。可以使用花括号解决上述二义性问题。比如，如果是第一种情况，可用${$a[1]}表示，如果是第二种情况，则可用${$a}[1]表示。

3.3.5 常量

常量是指在程序执行中无法修改的值。每个常量有一个名称，通过名称访问相应的常量值。在 PHP 中，常量通过 define 函数定义：

```
bool define(string $name, mixed $value [, bool $case_insensitive = false])
```

其中，$name 指定常量的名称，$value 指定常量的值。默认情况下，常量名是区分字母大小写的。若将$case_insensitive 设置为 true，则该常量名是不区分字母大小写的。

下面代码定义了一个名为 SECONDS 的常量，其值为一天包含的秒数，该常量名区分大小写。

```
define("SECONDS", 24*60*60);
```

常量的作用域是全局的，一旦定义就可以在脚本的任何位置引用，包括函数内或函数外。在作用域内，常量值不能修改，或重新定义。

常量名的命名规则与变量名一样，即以字母或下画线开头，后跟任意数量的字母、数字和下画线。默认情况下，常量名是大小写敏感的。按惯例，常量名一般采用大写字母。与变量不同，常量名前不需要加符号$。例如，下面代码引用了上面定义的常量：

```
echo SECONDS*10;
```

在 PHP 5.6 版本之前，常量值只能是标量数据，或者 NULL。标量数据包括整数、浮点数、字符串和布尔值。从 PHP 5.6 版本开始，常量值还可以是数组常量。

3.4 实例：创建动态水平导航栏

本书各章的实例将演示教务选课系统管理员子系统的开发。本实例创建的水平导航栏将作为模块用于子系统的各功能页面中。所谓动态是指它能根据输入参数使导航栏的当前菜单项醒目显示，并显示登录管理员的姓名。

先创建一个名为 xk 的 PHP 应用程序项目，然后在其“源文件”结点下新建名为 admin 和 css 的两个文件夹。admin 文件夹用于存放子系统的页面文件和模块文件，css 文件夹用

于存放子系统的外部样式表文件。

在 css 文件夹下创建一个名为 xk.css 的样式表文件。就本实例而言，该样式表文件只需包含两部分样式：基本样式和水平导航栏样式。这两部分样式在例 2-19 的外部样式表文件 navigation.css 中都已经定义，这里只需复制过来即可。

在 admin 文件夹下创建名为 navigation_admin.php 的 PHP 文件。下面是该 PHP 文件的代码，它是在例 2-19 中创建的页面文件 2-19.html 的基础上修改形成的。

```
1.  <!--
2.  *  功能：根据应用状态动态呈现导航栏
3.  *  输入：链入外部样式表<link rel='stylesheet' type='text/css' href='/xk/
           css/ xk.css' />
4.  *          $name：登录管理员的姓名
5.  *          $choice：当前页面对应的菜单项序号，取 1、2 或 3
6.  -->
7.  <!-- <link rel='stylesheet' type='text/css' href='/xk/css/xk.css' /> -->
8.  <?php
9.  // $name = "刘绍军";
10. // $choice = 1;
11. ?>
12. <div class='nav'>
13.    <a class='right' href='exit.php'>退出</a>
14.    <span class='right'><?php echo $name."，您好" ?></span>
15.    <a class='left <?php echo $choice===1 ? "current" : ""?>' href=
       'teacher_p.php'>
16.        浏览教师信息
17.    </a>
18.    <a class='left <?php echo $choice===2 ? "current" : ""?>' href='course_
       p.php'>添加课程</a>
19.    <a class='left <?php echo $choice===3 ? "current" : ""?>' href=
       'opencourse_p.php'>
20.        维护开课信息
21.    </a>
22.    <div style='clear: both'></div>
23. </div>
```

如果要单独运行和调试该文件，可以使用第 7~11 行之间被注释的代码。但作为一个模块，并不需要这些被注释的代码，链入外部样式表、设置变量（输入参数）值都应该在调用文件中完成。

用于浏览教师信息的页面文件 teacher_p.php、用于添加课程的页面文件 course_p.php 和用于维护开课信息的页面文件 opencourse_p.php 将会分别在第 8~10 章介绍。用于退出登录的 PHP 文件 exit.php 也将在第 9 章介绍。

习 题 3

1. 写出下面 PHP 代码运行的输出结果。

（1）

```
$a = -016;
$b = 0x16;
$c = $a + $b;
echo $c;
```

（2）

```
$x = 456;
$s = "abc\$\'123 $x 789";
echo $s;
```

（3）

```
$a = true;
$b = -$a;
$c = $b + "12.6p";
echo $c;
```

（4）

```
$a = -1;
$b = (bool)$a;
$c = $b + (int)"12.6p";
echo $c;
```

（5）

```
$a = "hello";
$b = &$a;
$b = 100;
echo $a;
```

（6）

```
$a="hello";
$b=&$a;
unset($b);
$b="world";
echo $a;
```

（7）

```
function get_count(){
```

```
    static $count = 2;
    return $count++;
}
$count = 5;
get_count();
echo get_count();
```

(8)

```
function get_count(){
    $count = 0;
    return ++$count;
}
$count = 5;
get_count();
echo get_count();
```

(9)

```
$GLOBALS['var1'] = 5;
$var2 = 1;
function get_value(){
    global $var2;
    $var1 = 0;
    return $var2++;
}
get_value();
echo $var1, $var2;
```

(10)

```
$str = "cd";
$$str = "landog";
$$str .= "ok";
echo $cd;
```

2. 简答题

(1) 什么是局部变量和全局变量？自定义函数内是否可以直接访问全局变量？
(2) 什么是静态变量？
(3) 其他类型数据转换为 boolean 类型时，哪些被认为是 false？
(4) empty 语言结构的功能是什么？在哪些情况下该语言结构返回 true？
(5) isset 语言结构的功能是什么？在哪些情况下该语言结构返回 true？

第 4 章　运算符与流程控制

本章主题：

- 运算符；
- 运算符的优先级与结合性；
- 流程控制语句；
- 包含文件。

运算符是一种可以对一个或多个操作数进行运算并产生结果的符号。运算符体现了一种语言所具有的对数据的直接处理能力。一般来说，一种运算符的操作数要有特定的数据类型，但很多情况下，PHP 会根据上下文自动将某种类型的数据转换成所需的类型。

程序的执行流程涉及程序的 3 种基本结构：顺序结构、选择结构和循环结构。从总体上看，程序是按顺序来执行代码块和函数体中的语句的，即按语句出现的先后次序依次执行语句，只有前面的语句执行完了才能执行后面的语句。选择结构是指在程序执行过程中，可以在两段代码中选择一段来执行，或者对一段代码是否执行进行选择。循环结构是指在程序执行过程中，可以对一段代码重复执行若干次。选择结构和循环结构都需要由一些特定的语句来实现。流程控制语句还包括跳转语句和包含文件语句。

本章首先介绍 PHP 的各种运算符及其使用，然后介绍控制流程的各种语句，包括实现选择结构的语句、实现循环结构的语句、跳转语句以及包含文件语句。

4.1　运　算　符

根据操作数的多少，运算符可分为单目运算符、双目运算符和三目运算符。只有一个操作数的运算符称为单目运算符，如果运算符出现在操作数之前，可称为前置运算符，如果运算符出现在操作数后面，可称为后置运算符。大多数运算符需要两个操作数且总是出现在两个操作数之间，称为双目运算符。需要 3 个操作数的运算符称为三目运算符，大多数编程语言支持三目条件运算符。

根据运算符的功能特点，PHP 运算符又可分为算术运算符、字符串运算符、比较运算符、逻辑运算符、位运算符、赋值运算符、三目条件运算符等。

4.1.1　算术运算符

算术运算符包括负号（–）、加（+）、减（–）、乘（*）、除（/）、求余（%）、指数运算（**）、增 1（++）和减 1（– –）等，如表 4-1 所示。

除法运算（/）一般返回浮点数。如果两个操作数都是整数（或字符串转换成的整数），并且正好能整除，此时它返回一个整数。

表 4-1　算术运算符

运算符	含义	例子
–	负号（单目）	–$x
+	加法（双目）	$x + $y
–	减法（双目）	$x – $y
*	乘法（双目）	$x * $y
/	除法（双目）	$x / $y
%	求余（双目）	$x % $y
**	指数运算（双目）	$x**$y（PHP 5.6 引入）
++	增 1（单目）	++$x，$x++
––	减 1（单目）	––$x，$x––

求余运算符（%）的操作数在运算之前都会被转换成整数（去除小数部分）。运算结果的符号（正负号）与被除数的符号相同，即$a%$b 的结果符号和$a 的符号相同。

在做算术运算时，如果出现非数值型操作数，那么 PHP 会尝试将其转换成数值型数据，然后再进行算术运算。如果转化不了，通常会产生致命错误（Fatal error）信息。

在 PHP 中，算术运算存在以下规则：

（1）进行算术除法运算（包括求余运算）时，除数不能为 0，否则将会产生警告（Warning）信息，运算结果为逻辑假（false）。

（2）在进行整数算术运算时，如果运算结果超出整型数的取值范围（溢出），那么运算结果自动转换成浮点型。

（3）在进行浮点数算术运算时，如果运算结果超出范围（上溢），则结果为无限值（正无穷或负无穷）。利用 is_infinite 函数可以测试一个值是否为无限值。

（4）浮点数运算的结果也可能是 NaN（Not a Number），例如两个无穷大相除、一个正无穷减去一个正无穷、0 乘以无穷大等。利用 is_nan 函数可以测试一个值是否为 NaN。

【例 4-1】 算术运算符的使用。代码如下：

```
<?php
// 除法运算
var_dump(12/4);
var_dump(12/5);
// 求余运算
$a = 12.8;
$b = 5.2;
echo $a % $b,' ', $a % -$b, ' ', -$a % $b, ' ', -$a % -$b, '<br/>';
// 指数运算
$r = 3;
echo M_PI*$r**2, "<br />";
// 增 1、减 1 运算
$x = 10;
echo $x++, ' ', $x, '<br/>';
```

```
15. $x = 10;
16. echo ++$x, ' ', $x, '<br/>';
17. ?>
```

下面是代码运行的输出结果：

```
int(3)
float(2.4)
2 2 -2 -2
28.274333882308
10 11
11 11
```

第 11 行代码用到了 PHP 的一个预定义常量 M_PI，可以返回圆周率的近似值。

4.1.2　字符串运算符

字符串运算符是指字符串连接运算符（.），用于将两个字符串连接成一个新的字符串返回。在做字符串连接运算时，如果出现非字符串操作数，那么 PHP 会尝试将其转换成字符串，然后再进行连接运算。

【例 4-2】 字符串运算符的使用。代码如下：

```
<?php
$a = "abc";
$b = "xyz";
echo $a." 123 ".$b."<br />";

$c = 123;
$d = 45.6;
echo $c+$d."<br />";
echo $c.$d."<br />";
?>
```

下面是代码运行的输出结果：

```
abc 123 xyz
168.6
12345.6
```

4.1.3　比较运算符

比较运算符用于比较两个数据的大小，其运算结果是布尔型的。比较运算符包括>、>=、<、<=、==、!=、===、!==等，如表 4-2 所示。

使用全等比较符（===）或不全等比较符（!==）比较两个数据时不涉及数据类型的转换，因为此时两个操作数的类型和值都要进行比较。仅当两个操作数的类型和值均相同，全等比较（===）才成立。只要两个操作数的类型不同，或者它们的值不相等，不全等比较（!==）就成立。

表 4-2　比较运算符

运算符	名　称	例　子
>	大于	$x > $y
>=	大于或等于	$x >= $y
<	小于	$x < $y
<=	小于或等于	$x <= $y
= =	相等	$x = = $y
!=或<>	不相等	$x != $y，$x <> $y
= = =	全等（类型和值均相同）	$x = = = $y
!= =	不全等	$x != = $y

当使用其他运算符比较两个数据时，如果两个操作数的类型不同，就会进行自动类型转换，使两个操作数具有相同类型，然后再进行比较。下面是数据比较运算时，有关数据类型转换的规则。

（1）如果一个操作数是 NULL，另一个操作数是字符串，则先将 NULL 转换成空串，然后再进行比较。

（2）如果一个操作数是 NULL，另一个操作数为非字符串，则两个操作数都转换成布尔型，然后再进行比较。这里，false<true。

（3）如果一个操作数是布尔型，则将另一个操作数也转换成布尔型，然后再进行比较。

（4）如果一个操作数是数值、另一个操作数是字符串，则字符串会被转换为数值，然后再进行比较。

（5）如果两个操作数都是数字内容的字符串（要求完全是数字内容，而非只是开始部分是数字内容），则将两个字符串都转换为数值，然后再按数值进行比较。

【例 4-3】 比较运算符的使用。代码如下：

```
<?php
var_dump(null < "0");        // true
var_dump(null < 0);          // false
var_dump(false < -1);        // true

var_dump(0 == "a");          // true
var_dump("1" == "01");       // true
var_dump("10" == "1e1");     // true
var_dump("10" == "10a");     // false

var_dump("1" === "01");      // false
var_dump("10" === "1e1");    // false
?>
```

4.1.4　逻辑运算符

逻辑运算符包括!、&&、||、xor、and 和 or。逻辑运算符的操作数应该是布尔型的，否

则会自动进行类型转化。逻辑运算的结果是布尔型的。逻辑运算符及其含义如表 4-3 所示。

表 4-3　逻辑运算符

运算符	含　　义	例　　子
!	逻辑非	!$x
&&	逻辑与	$x && $y
||	逻辑或	$x || $y
xor	逻辑异或	$x xor $y
and	逻辑与	$x and $y
or	逻辑或	$x or $y

逻辑非（!）是单目运算符，其运算结果与操作数的值正好相反。操作数为 true，结果为 false；操作数为 false，结果为 true。

逻辑与运算符（&&、and）具有“并且”的含义，只有当两个操作数的值均为 true 时，运算结果才为 true；否则，运算结果为 false。

逻辑或运算符（||、or）具有“或者”的含义，两个操作数中，只要有一个为 true，则运算结果就为 true；否则，运算结果为 false。

逻辑异或运算符（xor）具有“唯一”和“或者”两重含义，两个操作数中，当有一个为 true，而另一个不是 true 时，运算结果为 true；否则，运算结果为 false。

逻辑运算符&&和 and 都是快速逻辑与，逻辑运算符||和 or 都是快速逻辑或。即如果左操作数已经能决定运算结果，则右操作数将不再计算。它们之间的区别只是优先级不同，&&的优先级高于 and，||的优先级高于 or。

在 PHP 中，逻辑运算符 and、or 和 xor 的优先级比赋值运算符的还要低。

【例 4-4】 逻辑运算符的使用。代码如下：

```
<?php
$a = 10;
$b = 8;
$c = $a>=10 || ++$b>8;
var_dump($c);            // true
var_dump($b);            // 8

$d = $a>=10 && ++$b>9;
var_dump($d);            // false
var_dump($b);            // 9

$e = ($a>=10 and $b>9);
$f = $a>=10 and $b>9;
var_dump($e);            // false
var_dump($f);            // true
?>
```

第 4 行代码中，逻辑或（||）运算符的左操作数（$a>=10）的值是逻辑真 true，所以右操作数将不再计算，变量$b 的值仍为 8。

第 8 行代码中，逻辑与（&&）运算符的左右操作数依次计算，左操作数为逻辑真 true，右操作数为逻辑假 false，逻辑与的运算结果为 false。在计算右操作数时，变量$b 的值变为 9。

在第 13 行代码中，赋值运算符（=）的优先级高于逻辑与运算符（and）的，所以先做赋值（=）运算，结果变量$f 为逻辑真 true，然后再做逻辑与（and）运算，但运算结果没有保存，被丢弃了。

4.1.5 位运算符

位运算符包括~、&、|、^、<<、>>，如表 4-4 所示。位运算符的操作数应该是整型的，否则会自动进行类型转换。位运算符将一个整型值当作一系列的二进制位来处理。位运算的结果是整型的。

表 4-4 位运算符

运算符	含 义	例 子
~	按位取反（单目）	~$x
&	按位与	$x & $y
|	按位或	$x | $y
^	按位异或	$x ^ $y
<<	左位移	$x << $y
>>	右位移	$x >> $y

前面 4 个运算符（~、&、|、^）也称为位逻辑运算符。原则上，位逻辑运算与前面介绍的逻辑运算的运算规则是相同的，只是逻辑运算符的运算对象是布尔型数据（true 或 false），而位逻辑运算符的运算对象是操作数（整数）内部的二进制位数据（0 或 1）。

需要注意的是，如果左右操作数都是字符串，则位逻辑运算符将对字符的 ASCII 值进行操作。

后面两个运算符（<<、>>）也称为位移运算符。向任何方向移出去的位都被丢弃。左移时右侧以零填充，符号位被移走意味着正负号不被保留。右移时左侧以符号位填充，意味着正负号被保留。

【例 4-5】 位运算符的使用。代码如下：

```
<?php
$a = -10;
$b = 10;
echo ~$a, " ", $a&$b, " ", $a|$b, " ", $a^$b, "<br />";

$x = -2;
$y = 2;
echo $x<<$y, " ", $x>>$y;
?>
```

下面是代码运行的输出结果：

```
9 2 -2 -4
-8 -1
```

其中第 4 行代码涉及 4 个位逻辑运算，第 8 行代码涉及 2 个位移运算。为使运算过程更加清晰，下面给出了各操作数及运算结果的内部二进制表示。这里假定整数采用 32 位的二进制补码表示。

```
$a:-10          11111111111111111111111111110110
$b:10           00000000000000000000000000001010
~$a:9           00000000000000000000000000001001
$a&$b:2         00000000000000000000000000000010
$a|$b:-2        11111111111111111111111111111110
$a^$b:-4        11111111111111111111111111111100
$x:-2           11111111111111111111111111111110
$y:2            00000000000000000000000000000010
$x<<$y:-8       11111111111111111111111111111000
$x>>$y:-1       11111111111111111111111111111111
```

4.1.6 赋值运算符

赋值运算符包括简单赋值运算符和组合赋值运算符。

1. 简单赋值运算符

简单赋值运算符（=）的语法格式如下：

```
<变量>=<表达式>
```

其功能是计算赋值运算符右端表达式的值，并将其赋给运算符左端变量。该值同时作为整个赋值运算表达式的运算结果。

2. 组合赋值运算符

组合赋值运算符（<op>=）将计算和赋值两种功能组合在一起，其语法格式如下：

```
<变量> <op>= <表达式>
```

其功能是将变量的值与表达式的值按指定的运算符 op 进行计算，然后再把计算的结果重新赋给变量。这里 op 包括双目的算术运算符、字符串运算符、位运算符以及数组联合运算符（+）。数组联合运算符将在第 9 章介绍。

【例 4-6】 赋值运算符的使用。代码如下：

```
<?php
$a = ($b = 4) + 5;
echo $a, " ", $b, "<br />";

$x = 3;
```

```
$x += 5;
$y = "Hello ";
$y .= "There!";
echo $x, " ", $y;
?>
```

下面是代码运行的输出结果：

```
9 4
8 Hello There!
```

4.1.7 其他运算符

这里介绍三目条件运算符?:、错误抑制运算符@和类型运算符 instanceof。

1. 三目条件运算符

三目条件运算符? :的操作数有 3 个，其语法格式如下：

```
<op1>?<op2>:<op3>
```

这里，op1 应该是布尔型的，否则会自动进行类型转换。如果 op1 为 true，则 op2 作为表达式的结果；如果 op1 为 false，则 op3 作为表达式的结果。这里，op2 和 op3 只选择其一进行计算。

【例 4-7】 三目条件运算符的使用。代码如下：

```
1.  <?php
2.  $a = 1;
3.  $b = "apple";
4.  $c = true ? $b : ++$a;
5.  echo $c," ", $a, "<br />";
6.  echo (true ? 'true' : false ? 't' : 'f');
7.  ?>
```

下面是代码运行的输出结果：

```
apple 1
t
```

第 4 行代码的三目条件运算表达式中，第 1 个操作数的值为 true，所以只计算第 2 个操作数$b 的值作为表达式的值，第 3 个操作数++$a 并没有计算，所以变量$a 的值并没有变化。

第 6 行代码涉及三目条件运算符的嵌套。在 PHP 中，三目条件运算符是左结合的，所以整个表达式的值为't '，而不是 true。

2. 错误抑制运算符

错误抑制运算符@可以放置在任何表达式之前，用以抑制错误消息，即计算表达式时产生的任何错误消息将被忽略。

在 PHP 中，主要有以下 3 种错误类型，即注意（Notices）、警告（Warnings）和致命错

误（Fatal errors）。当出现注意和警告信息时，程序仍将继续执行，但当发生致命错误时，将导致程序终止运行。错误抑制运算符只用于抑制在计算表达式时出现的各种错误信息，并不会影响程序的运行行为，即该继续执行的还会继续执行，该终止执行的仍将终止执行。

错误抑制运算符只对表达式有效。一般来说，如果能从某处得到值，就可以将@放置在它之前。比如，可以将@放置在变量、常量、函数调用、include 调用之前。但不能把它放在函数或类的定义之前，也不能用于 if 和 foreach 等控制流程的语句。

3. 类型运算符

类型运算符 instanceof 用于检测一个对象是否为某个特定类或其子类的实例，其语法格式如下：

```
<op1> instanceof <op2>
```

其中，op1 应该是一个对象，op2 表示某个类类型或接口类型。如果对象 op1 是类 op2 或其子类的实例，或者是实现接口 op2 的某个类的实例，则表达式返回 true，否则返回 false。

【例 4-8】 类型运算符的使用。代码如下：

```php
<?php
class SuperClass {}
class SubClass extends SuperClass {}
class OtherClass {}

$o = new SubClass();

var_dump($o instanceof SubClass);
var_dump($o instanceof SuperClass);
var_dump($o instanceof OtherClass);
?>
```

下面是代码运行的结果：

```
bool(true)
bool(true)
bool(false)
```

这里，SuperClass 是 SubClass 的超类，而 OtherClass 不是 SubClass 的超类。SubClass 类的实例可以被看作是 SuperClass 类的对象，但不能被看作是 OtherClass 类的对象。

4.2 表 达 式

表达式是由运算符和操作数按一定的语法规则连接起来的式子。前面已经对运算符做了较为全面的介绍，包括算术运算符、关系运算符、逻辑运算符、位运算符、三目条件运算符以及赋值运算符等。操作数包括文字、常量、变量、函数调用、数组元素访问等。

这里，各种操作数本身也是表达式。所以表达式的这个定义是递归的，即在表达式的定义中又用到了表达式自身。

当一个表达式需要使用许多不同的运算符时，搞清楚 PHP 解释器对表达式的计算次序就显得非常重要。例如表达式 x + 5 – (x = 10) + x * 2，是先计算 x + 5 还是先计算 x * 2？其中的 x 是取原先的值还是取 10？

运算符的优先级和结合性影响着表达式的计算次序。当一个操作数同时为前后两个运算符的操作数时，如果前面运算符的优先级高于后面运算符的优先级，那么该操作数作为前面运算符的操作数先进行运算，运算结果作为后面运算符的操作数；如果后面运算符的优先级高于前面运算符的优先级，那么该操作数作为后面运算符的操作数先进行运算，运算结果作为前面运算符的操作数。如果前后两个运算符的优先级相同，那么要看运算符的结合性。如果是左结合（从左到右），那么该操作数作为前面运算符的操作数先进行运算，运算结果作为后面运算符的操作数；如果是右结合（从右到左），那么该操作数作为后面运算符的操作数先进行运算，运算结果作为前面运算符的操作数。例如：

```
x + y * z                相当于 x + (y * z)            *优先于+
x = y - z                相当于 x = (y - z)            -优先于=
x + y - z                相当于 (x + y) - z            同级左结合
x = y = 13               相当于 x = (y = 13)           同级右结合
a ? b : c ? d : e        相当于 (a ? b : c) ? d : e    同级左结合
```

表 4-5 列出了 PHP 运算符的优先级和结合性。其中，各运算符按优先级降序排列，优先级为 1 的运算符优先级最高，同一行中的运算符的优先级相同。

表 4-5　运算符的优先级和结合性

优先级	运算符	结合性
1	new	—
2	[]	左
3	**	右
4	++、– –、~、(int)、(float)、(string)、(array)、(object)、(bool)、@	右
5	instanceof	—
6	!	右
7	*、/、%	左
8	+、–、.	左
9	<<、>>	左
10	<、<=、>、>=	—
11	==、!=（<>）、===、!==	—
12	&	左
13	^	左
14	|	左
15	&&	左
16	||	左

续表

优先级	运算符	结合性
17	? :	左
18	=、+=、–=、*=、/=、%=、.=、&=、|=、^=、<<=、>>=	右
19	and	左
20	xor	左
21	or	左
22	,	左

从表 4-5 中可以发现，有些运算符没有规定结合性，如比较运算符等。当出现前后两个运算符具有相同的优先级但没有规定结合性时，将无法确定它们计算的先后次序，所以这种情况是非法的。比如表达式 1<$x<=10 是非法的，因为运算符<和<=的优先级相同但没有规定结合性。而表达式 1 <= $x == 1 是合法的，虽然运算符<=和==都没有规定结合性，但它们的优先级不同。

总的来说，表达式的计算按从左到右、并尊重运算符的优先级和结合性的原则进行。表达式在从左到右的计算过程中，当前运算符的操作数首先被计算（左操作数先于右操作数被计算），然后再判断当前运算是马上执行还是暂缓执行。如果当前运算符的优先级高于后面运算符的优先级，或者前后两个运算符优先级相同但为左结合，那么当前运算就可以马上执行，运算结果将作为新的操作数参与表达式接下来的计算过程。如果当前运算符的优先级低于后面运算符的优先级，或者前后两个运算符优先级相同但为右结合，那么暂缓执行当前运算，转而考虑下一个运算。

另外，有些操作数本身就是一个由圆括号括起来的子表达式，对这些操作数的计算同样需要遵循上述原则和过程。下面通过例子说明表达式的计算过程。

假设变量$x 已经由下面语句定义：

```
$x = 15;
```

表达式$x + 5 – ($x = 10) + $x * 2 的计算过程如下：

步骤 1：计算加法运算，表达式变为“20 – ($x = 10) + $x * 2”。

步骤 2：计算子表达式，表达式变为“20 – 10 + $x * 2”，其中变量$x 的值变为 10。

步骤 3：计算减法运算，表达式变为“10 + $x * 2”。

步骤 4：计算乘法运算，表达式变为“10 + 20”。

步骤 5：计算加法运算，得到表达式的计算结果为整数 30。

这里，步骤 1 计算加法运算时，变量$x 取原先的值 15。步骤 2 计算子表达式，其计算过程与一般表达式的计算过程相同。该子表达式的计算除了返回一个结果值 10 之外，还产生了一个副作用，即将变量$x 置为 10。所以，后面再访问变量$x 时将使用该新值。

【例 4-9】 优先级与结合性的应用。代码如下：

```
<?php
$a = 10; $b = 20; $c = 30;
echo "$a+$b=",$a+$b;
```

```
4.  echo "<br />";
5.  echo "$a+$b=".$a+$b;
6.  echo "<br />";
7.  $d = $a>=10 || $b++>20 && ++$c>30;
8.  var_dump($d, $b, $c);
9.  $d = $b++>20 && ++$c>30 || $a>=10;
10. var_dump($d, $b, $c);
11. ?>
```

下面是代码运行的输出结果：

```
10+20=30
30
bool(true) int(20) int(30)
bool(true) int(21) int(30)
```

这里，第 3 行代码依次输出两个表达式的值，如果把逗号（,）看作是运算符，那么它的优先级是最低的。第 5 行代码用到字符串连接运算符（.）和算术加运算符（+），两者优先级相同，采用左结合，所以先做连接运算，结果为“10+20=10”，然后再做加运算，结果为 30。

第 7 行代码中的逻辑运算表达式，运算符&&的优先级比运算符||的高，表达式相当于

```
$a>=10 || ($b++>20 && ++$c>30)
```

因为||是快速逻辑或，所以先计算左操作数$a>=10。由于其结果为 true，所以右操作数不再计算。

第 9 行代码中的逻辑运算表达式相当于

```
($b++>20 && ++$c>30) || $a>=10
```

首先计算快速逻辑与（&&）的左操作数$b++>20，结果为 false，变量$b 的值增 1 变为 21。由于快速逻辑与（&&）的左操作数为 false，所以右操作数不再计算，圆括号内的子表达式的结果为 false。最后计算快速逻辑或（||）的右操作数$a>=10，结果为 true，所以整个表达式的结果为 true。

4.3 流程控制

本节介绍控制程序执行流程的相关语句，包括支持选择结构的语句、支持循环结构的语句以及若干跳转语句。

4.3.1 语句与语句块

PHP 脚本是语句的集合。一条 PHP 语句可以是赋值语句、表达式语句、条件语句、循环语句、函数调用等，甚至可以是空语句。一条语句以分号（;）结尾，分号是 PHP 语句的终止符号。

所谓空语句就是仅包含一个分号、不执行任何操作的语句，用于程序中某处语法上要求应该有一条语句但实际不需要做任何数据处理的情况。比如：

```
for($i = 1; $i <= 10000; $i++) ;
```

该循环语句的循环体只包含一条空语句。虽然循环体被循环执行了 10000 次，但并不做任何数据处理。

为便于阅读，通常一条语句写一行。但从语法上说，一行也可以包含多条语句，一条语句可以跨越多行。下面代码段 1 中，一行包含两条赋值语句，代码段 2 中，一条输出语句跨越两行，这些都是允许的。

代码段 1：

```
$x = 0; $y = 2;
```

代码段 2：

```
echo $x, "hello ",
$y, "world!";
```

由花括号（{ }）括起来的一组语句称为块语句。块语句内的各语句都须以分号结尾。块语句本身也是一条语句，但一般不需要以分号结尾。需要注意的是，在 PHP 中，块语句并不是一个新的变量作用域范围。比如下面代码：

```
$x = 0;
{
$x++;
$y = $x + 10;
}
echo $x, $y;
```

这里，变量$x 是在块语句前建立的，在之后的代码（包括块语句）中都是有效的。变量$y 是在块语句中建立，但在离开块语句后，仍然是可用的。

4.3.2 选择结构

在 PHP 中，支持选择结构的语句包括 if、if…else、if…elseif…else 和 switch 语句。

1. if 语句

if 语句的语法格式如下：

```
if(<expr>) <statement>
```

其中，expr 应该是布尔型的表达式，否则会自动转换成布尔型。statement 既可以是简单语句，也可以是块语句。若 expr 的值为 true，则执行 statement 语句；若 expr 的值为 false，则跳过 statement 语句，直接执行该 if 语句后面的语句。图 4-1 是该语句执行流程的图示说明。

【例 4-10】 使用 if 语句。编写函数 f_max，其功能是接收两个数值$x 和$y，然后返回其中的较大值。代码如下：

```
<?php
function f_max($x, $y) {
   $max = $x;
   if($y>$max) $max = $y;
   return $max;
}
echo f_max(3, 8);
?>
```

本章有些例子会使用函数，但目的还是为了介绍相关语句的使用。有关函数的详细内容将在第 5 章介绍。

2．if…else 语句

if…else 语句的语法格式如下：

```
if(<expr>) <statement1> else <statement2>
```

其中，表达式 expr 应该是布尔型的，否则会自动转换成布尔型。statement1 和 statement2 既可以是简单语句，也可以是块语句。语句根据 expr 的值从 statement1 和 statement2 中选择一条执行。若 expr 的值为 true，则执行 statement1；若 expr 的值为 false，则执行 statement2。然后转入下一条语句执行。图 4-2 是该语句执行流程的图示说明。

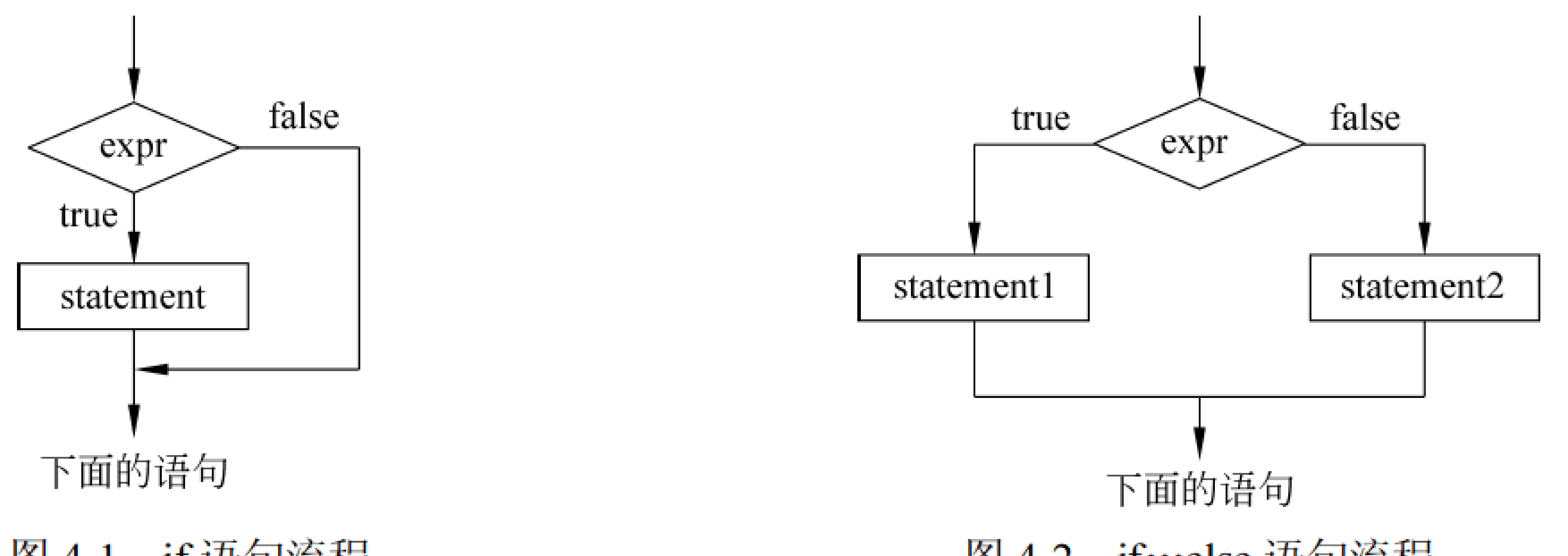


图 4-1　if 语句流程　　　　图 4-2　if…else 语句流程

【例 4-11】 使用 if…else 语句。编写函数 f_square，其功能是接收两个整型值$x 和$y，然后返回较大值的平方。代码如下：

```
<?php
function f_max($x, $y) {
   if($x>$y) {
      $max = $x;
   } else {
      $max = $y;
   }
   return $max * $max;
}
echo f_max(3,8);
?>
```

3. if…elseif…else 语句

if…elseif…else 语句的语法格式如下：

```
if(<expr_1>) <statement_1>
elseif(<expr_2>) <statement_2>
...
elseif(<expr_n>) <statement_n>
else <statement_n+1>
```

本质上，该语句就是 if…else 语句的嵌套，即在 if 表达式值为 false 时再执行嵌套的 if…else 语句。该语句用于实现多分支选择结构。图 4-3 是该语句执行流程的图示说明。

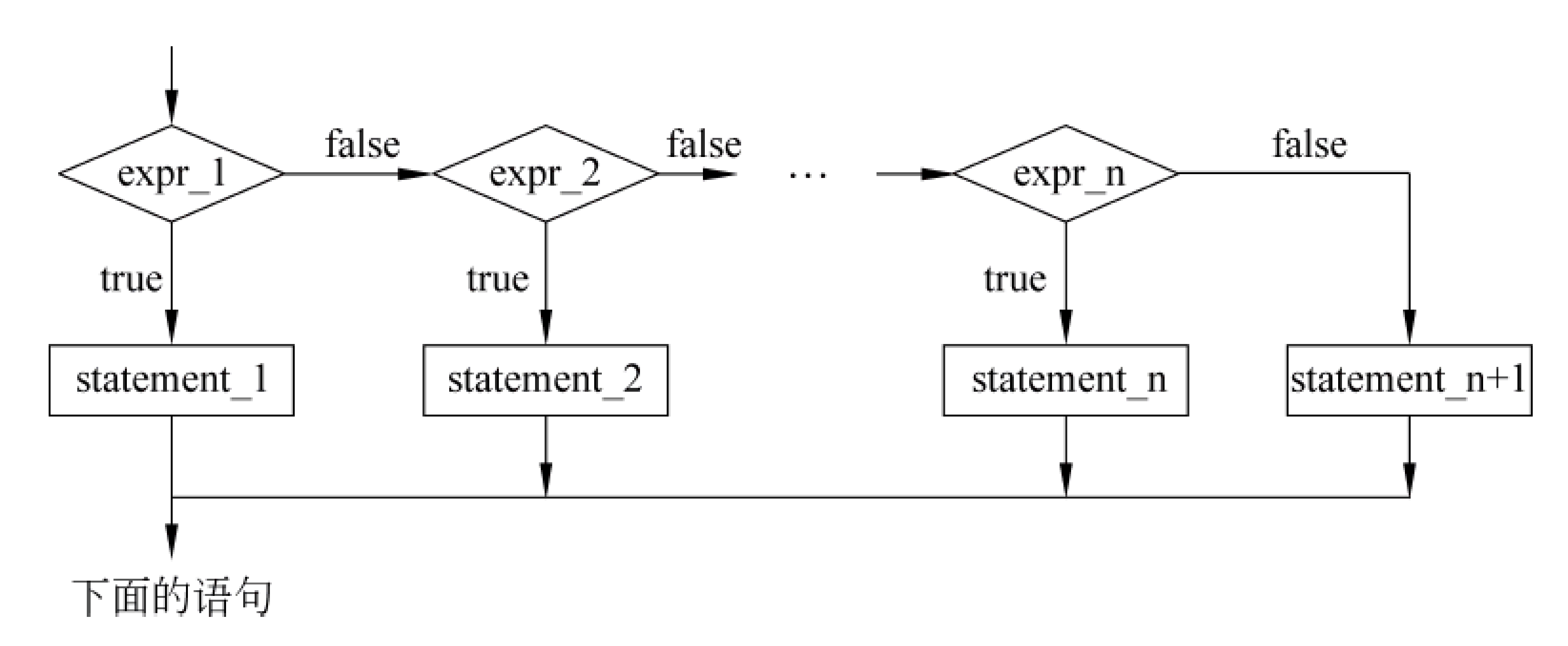


图 4-3　if…elseif…else 语句流程

该语句执行时，如果 expr_1 的值为 true，执行 statement_1，然后执行该 if 语句的下一条语句。表达式 expr_i 只在前面所有的条件表达式（expr_1、expr_2、…）的值均为 false 时才计算，如果 expr_i 的值为 true，语句 statement_i 被执行，然后执行该 if 语句的下一条语句。else <statement_n+1>是可选的，语句 statement_n+1 只在前面所有的条件表达式的值都为 false 时才被执行，然后执行下一条语句。

【例 4-12】 使用 if…elseif…else 语句。判断$x 的值是小于 0、等于 0 或大于 0，分别输出-1、0 或 1。代码如下：

```
<?php
$x = 2;
if($x==0) {
   $sign = 0;
} elseif($x>0) {
   $sign = 1;
} else {
   $sign = -1;
}
echo $sign;
?>
```

4. switch 语句

一般来说，多分支选择结构既可以用 if…else 语句的嵌套或 if…elseif…else 语句来实现，也可以用 switch 语句来实现。switch 语句的语法格式如下：

```
switch(<expr>) {
   case <expr_1>: [<statements_1>] [break;]
   case <expr_2>: [<statements_2>] [break;]
   …
   case <expr_n>: [<statements_n>] [break;]
   [default: <statements_n+1> [break;]]
}
```

switch 语句中的表达式只能是标量类型的表达式。语句执行时，首先计算 switch 后面的表达式（expr）的值，然后将其与各 case 后面的表达式（expr_1、expr_2、…）的值依次进行相等比较（==）：

（1）若发现某个 case 后面的表达式值与表达式 expr 的值相等，就转入该 case 标号后面的第一条语句继续往下执行。当执行到 break 语句时，跳出 switch 语句，接着执行 switch 语句后面的语句。

（2）如果没有一个 case 后面的表达式值与表达式 expr 的值相等，就转入 default 标号后面的第一条语句继续往下执行。当执行到 break 语句时，跳出 switch 语句，接着执行 switch 语句后面的语句。

（3）如果没有一个 case 后面的表达式值与表达式 expr 的值相等，且不存在 default 部分，则程序跳出 switch 语句，直接执行 switch 语句后面的语句。

【例 4-13】 使用 switch 语句。该示例代码的功能是根据月份值（$month）输出季度名称。代码如下：

```
<?php
$month = 9;
switch($month) {
   case 1:
   case 2:
   case 3: $s="first quarter"; break;
   case 4:
   case 5:
   case 6: $s="second quarter"; break;
   case 7:
   case 8:
   case 9: $s="third quarter"; break;
   case 10:
   case 11:
   case 12: $s="forth quarter"; break;
   default: $s="error data";
}
echo $s;
?>
```

4.3.3 循环结构

在 PHP 中，实现循环结构的语句包括 while、do…while、for 和 foreach 语句。

1．while 语句

while 循环语句的语法格式如下：

```
while(<expr>) <statement>
```

其中，表达式 expr 应该是布尔型的，否则会自动转换成布尔型。当语句执行时，首先计算 expr 的值，如果 expr 的值为 true，则执行其内部嵌套的 statement。执行完 statement 后，程序再次计算并判断 expr 的值是否为 true。这个过程循环进行，直到 expr 的值为 false 时，跳出该 while 语句，执行后面的语句。while 语句的执行流程如图 4-4 所示。

就如 while 语句，所有循环语句的内部都嵌套着一个语句，该嵌套语句通常被称作为循环体。循环体可以是单条语句，也可以是块语句。从总体上看，循环语句的作用就是控制循环体被重复执行一定的次数。

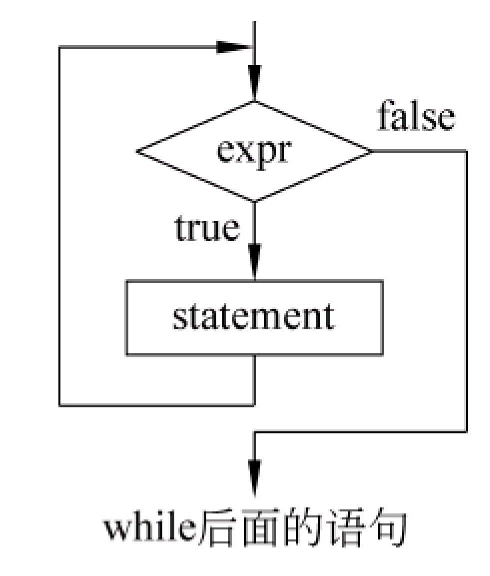


图 4-4　while 语句执行流程

while 语句也被称为“当型”循环语句，其特点是：循环体可能被执行多次，也可能一次也不执行。

【例 4-14】 使用 while 语句。函数 f_factorial 的功能是计算$n 的阶乘，即$n!。代码如下：

```
<?php
function f_factorial($n) {
   if($n<0) return null;
   if($n==0 || $n==1) return 1;
   $p = 1;
   $i = 2;
   while($i<=$n) {
      $p *= $i;
      $i++;
   }
   return $p;
}
echo f_factorial(5);
?>
```

2．do…while 语句

do…while 循环语句的语法格式如下：

```
do <statement> while(<expr>);
```

由于被嵌套的 statement 并不出现在 do 语句的末尾，所以格式特别强调 do 语句本身的

末尾应该有一个分号。

语句中表达式 expr 应该是布尔型的，否则会自动转换成布尔型。当语句执行时，首先执行作为循环体的 statement，然后计算 expr 的值。如果 expr 的值为 true，则再次执行 statement。这个过程循环进行，直到 expr 的值为 false 时，跳出该 do-while 语句。do-while 语句的执行流程如图 4-5 所示。

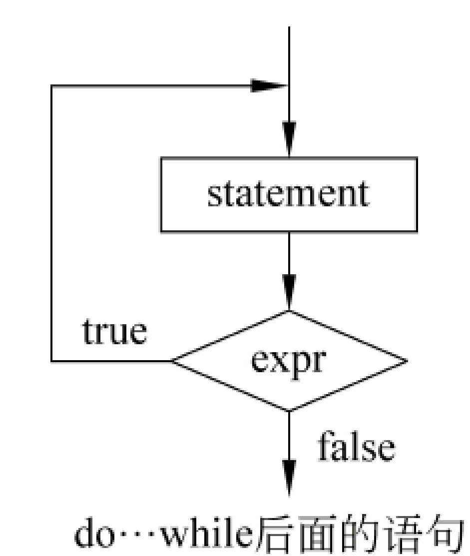


图 4-5　do…while 语句执行流程

do…while 语句也被称为“直到型”循环语句，其特点是：作为循环体的 statement 至少执行一次。

3. for 语句

for 循环语句的语法格式如下：

```
for([<expr1>]; [<expr2>]; [<expr3>]) <statement>
```

其中，expr1、expr2 和 expr3 都可以是空的，此时 expr2 的值默认为 true。expr1、expr2 和 expr3 也可以包含多个表达式，此时各表达式之间用逗号（,）分隔。

当语句执行时，首先计算表达式 expr1，然后计算表达式 expr2。如果 expr2 的值为 true，则执行循环体 statement，然后计算表达式 expr3，最后再次计算并判断 expr2 的值是否为 true。这个过程重复进行，直到 expr2 的值为 false 时，跳出该 for 循环语句。

如果 expr2 包含多个表达式，则每次执行循环体 statement 前，每个表达式都会被计算，但以其中最后一个表达式的值作为是否执行循环体 statement 的依据。

【例 4-15】 使用 for 语句。编写函数 f_sum，实现计算 1! + 2! +…+ $n! 的功能。代码如下：

```
<?php
function f_sum($n) {
   $s = 0;
   for($i = 1,$p = 1; $i <= $n; $i++) {
      $p = $p * $i;
      $s = $s + $p;
   }
   return $s;
}
echo f_sum(5);
?>
```

4. foreach 语句

foreach 循环语句提供了一种遍历数组（或对象）的便捷手段，它包含两种格式。

格式 1：

```
foreach(<array_expression> as <$value>) <statement>
```

格式 2：

```
foreach(<array_expression> as <$key>=><$value>) <statement>
```

语句执行时，指定数组的各元素会依次赋给指定的变量，并执行循环体 statement。循环体 statement 可以对当前数组元素进行所需的处理。

在格式 1 中，每次只是把当前元素的值赋给变量$value，而在格式 2 中，每次会把当前元素的键和值分别赋给变量$key 和变量$value。

如果不只是要访问数组元素的值，而是要修改数组元素的值，可以在接收元素值的变量$value 前加上&。这样将实现引用赋值，而不是值赋值。

【例 4-16】 使用 foreach 语句。代码如下：

```
<?php
$a = array(1, 2, 3, 4, 5);
$s=0;
foreach($a as $v) {      // 数组元素值赋值，实现累加
    $s += $v;
}
echo $s, "<br />";

foreach($a as &$v) {     // 数组元素引用赋值，改变元素值
    $v *= $v;
}
print_r($a);
?>
```

下面是代码运行的输出结果：

```
15
Array ( [0] => 1 [1] => 4 [2] => 9 [3] => 16 [4] => 25 )
```

该例代码用到了 print_r 函数。该函数可以输出表达式的信息，尤其适合于输出一个数组的信息，此时它会输出数组每个元素的键和值。

4.3.4 跳转语句

跳转语句包括 return 语句、exit 语句、break 语句和 continue 语句。

1．return 语句

return 语句的语法格式如下：

```
return [<expr>];
```

return 经常出现在函数内。一个函数中可以包含多个 return 语句，程序一旦执行到 return 语句就会结束函数的执行、返回到函数的调用处。如果一个函数需要返回值，那么该函数内的 return 语句就应该带表达式 expr。这样，当执行到 return 语句返回时，表达式 expr 被计算，其值作为函数值返回给调用者。

一般情况下，出现在函数体末尾不带表达式的 return 语句可以被省略。

return 语句也可以出现在脚本文件中。如果 return 语句出现在被包含文件中，那么执行 return 语句将结束被包含文件的执行，返回到包含文件继续执行。如果在主脚本文件中调用

return 语句，则将结束整个脚本的执行。

2．exit 语句

exit 是一种语言结构，但也可以像函数一样使用，其语法格式如下：

格式 1：

```
void exit;
```

格式 2：

```
void exit([string <$status>]);
```

exit 用于终止脚本的执行。如果包含参数字符串$status，那么退出脚本执行之前会输出该字符串。与 return 不同，exit 即使出现在函数或被包含文件中，它也是终止脚本执行并退出，而不会返回到函数调用处或包含文件继续执行。

下面代码是一条表达式语句，其中逻辑或运算符 or 的优先级比赋值运算符=的要低。

```
$sn = @$_SESSION["sn"] or exit("还没有注册！");
```

语句执行时，如果已经注册了名为 sn 的会话变量，且其值不为 0、""、"0"或 false，那么就会跳过 exit，继续执行下面的代码；否则，执行 exit 语句，输出相应的信息，然后终止脚本的执行。

3．break 语句

break 语句的语法格式如下：

```
break [<n>];
```

break 语句应该出现在循环语句（for、foreach、while、do-while）或开关语句（switch）内，其功能是结束包含它的某个循环语句或开关语句的执行。

可选项 n 应该取数值常量（如 1，2，3 等），用于指明跳出包含该 break 语句的第几层循环语句或开关语句。如果缺省该选项，n 的值为 1，此时 break 语句跳出包含它的最内层循环语句或开关语句。

【例 4-17】 使用 break 语句。代码如下：

```
<?php
$s = 0;
$i = 0;
while (++$i < 5) {
   $j = 0;
   while (++$j <= $i) {
      if ($j * $j > $i + 3) break;
      $s = $s + $j;
   }
   $s = $s + $i;
}
echo '$s = ' . $s;
```

```
?>
```

下面是代码运行的输出结果：

```
$s = 20
```

程序运行过程中，当条件$j*$j>$i+3 成立时，程序将跳出内层循环语句。这样，当前的$j 值不会被累加到$s 上、后面的$j 也不会被考虑。

4．continue 语句

continue 语句的语法格式如下：

```
continue [<n>];
```

通常，continue 语句出现在循环语句（for、foreach、while、do…while）中，其功能是结束循环体的本次执行，准备进入循环体的下次执行。

continue 语句也可以出现在开关语句（switch）中，此时它的功能与 break 语句一样，用以结束开关语句的执行。

可选项 n 应该取数值常量（取 1，2，3 等），用于指明该语句作用的是包含它的第几层循环语句或开关语句。比如 n 的值取 3，而包含 continue 语句的从内到外依次有 switch 语句、for 语句和 switch 语句，那么该 continue 语句作用的对象是外层的 switch 语句，所以会结束外层的 switch 语句的执行。当缺省该选项时，n 的值取 1。

【例 4-18】 continue 语句的使用。代码如下：

```
<?php
$s = 0;
$i = 0;
while (++$i < 5) {
   $j = 0;
   while (++$j <= $i) {
      if ($j * $j > $i + 3) continue 2;
      $s = $s + $j;
   }
   $s = $s + $i;
}
echo '$s = ' . $s;
?>
```

下面是代码运行的输出结果：

```
$s = 13
```

程序运行过程中，当条件$j*$j>$i+3 成立时，程序将结束外循环体的本次执行。这样，不仅当前的$j 值不会被累加到$s 上、后面的$j 不会被考虑，而且当前的$i 值也不会被累加到$s 上。

4.4 包 含 文 件

PHP 脚本代码中可以嵌入包含文件语句。当执行到包含文件语句时，PHP 解释器将包含并处理执行另一个文件的代码。包含文件语句可以出现在一般的脚本中，也可以出现在用户自定义函数内。被包含文件可以是 PHP 文件，也可以是 HTML 文件。

4.4.1 包含文件语句

包含文件语句包括 include、require、include_once 和 require_once 语句。

1. include 语句

include 语句的语法格式如下：

```
include <filespec>;
```

或

```
include(<filespec>);
```

其功能是包含并处理由 filespec 指定的文件。被包含文件代码的变量作用域继承该语句所在处的变量作用域，即此处可用的变量在被包含文件中也可以使用。反过来，被包含文件中定义的函数、变量等在包含文件的后续代码中也可以使用。

若指定文件不存在，语句将产生一个警告（Warning）信息，但代码会继续往下运行。

与有些语言结构（如 isset、print）等类似，包含文件语句也有返回值。在默认情况下，如果包含文件操作成功执行，那么 include 语句返回整数 1；如果指定的被包含文件不存在，那么 include 语句除了产生一个警告（Warning）信息外，返回布尔值 false。在被包含文件通过 return 语句返回的情况下，如果 return 语句不带表达式，那么 include 语句返回 NULL 值；如果 return 语句带表达式，那么 include 语句返回该表达式的值。

2. require 语句

require 语句的语法格式如下：

```
require <filespec>;
```

或

```
require(<filespec>);
```

与 include 语句相同，require 语句也用于包含并处理由 filespec 指定的文件。两者唯一的不同是，若指定的文件不存在，require 语句产生一个致命错误（Fatal error）信息，代码不再继续往下执行。

3. include_once 语句

可以多次使用 include 或 require 语句包含同一个文件。但如果被包含文件中存在函数定义时，那么在第 2 次用 include 或 require 语句包含该文件时，将出现函数重复定义的情况。此时，将产生一个致命错误（Fatal error）信息，代码不再往下执行。使用 include_once 或

require_once 语句可以避免这种错误的出现。

include_once 语句的语法格式如下：

```
include_once <filespec>;
```

或

```
include_once(<filespec>);
```

与 include 语句类似，include_once 语句的功能也是包含并处理指定的文件。两者唯一的区别是，如果之前已经包含了指定文件，include_once 语句将不会再次包含，并且返回布尔值 true。

4．require_once 语句

require_once 语句的语法格式如下：

```
require_once <filespec>;
```

或

```
require_once(<filespec>);
```

与 require 语句类似，require_once 语句的功能也是包含并处理指定的文件。两者唯一的区别是，如果之前已经包含了指定文件，require_once 语句将不会再次包含，并且返回布尔值 true。

在 PHP 中，包含文件技术与函数的作用类似，都是数据处理模块化和程序代码重用的一项技术，也是页面模块化和页面代码重用的一种手段。比如可以把一个完整的页面分割成几个相对独立的部分，并各自由相应的 PHP 文件来实现，再由一个主 PHP 文件把这些实现页面部分内容的 PHP 文件包含进来，组合形成一个完整的页面。这里主 PHP 文件是包含文件，也称为调用文件；实现页面部分内容的 PHP 文件是被包含文件，也称为被调用文件。

相对于包含文件技术，函数具有更好的封装性，并且更方便组织和维护。本书将在第 5 章介绍函数的定义、调用等相关技术。

【例 4-19】 使用包含文件语句。本例涉及 3 个文件：一个是 include.php，作为包含文件或调用文件；另一个是 included1.php，作为被包含文件或被调用文件，在一般脚本中被调用；最后一个是 included2.php，作为被包含文件或被调用文件，在用户自定义函数内被调用。

包含文件 include.php 代码如下：

```
<?php
$color = 'yellow';
$fruit = 'banana';
include 'included1.php';
echo "A $color $fruit <br/>";
myfunction();
echo "A $color $fruit <br/>";
function myfunction() {
```

```
9.     $color = 'yellow';
10.    $fruit = 'banana';
11.    include 'included2.php';
12.    echo "A $color $fruit <br/>";
13. }
14. ?>
```

被包含文件 included1.php 代码如下：

```
<?php
echo "A $color $fruit <br/>";
$color = 'green';
$fruit = 'apple';
?>
```

被包含文件 included2.php：

```
<?php
global $color;
echo "A $color $fruit <br/>";
$color = 'purple';
$fruit = 'grape';
?>
```

当请求 include.php 时，代码运行的输出结果如下：

```
A yellow banana
A green apple
A green banana
A purple grape
A purple apple
```

这里对自定义函数内调用被包含文件的情况进行说明。在主文件第 6 行调用自定义函数前，有两个全局变量：变量$color 的值为'green'，变量$fruit 的值为'apple'。在执行函数体时，首先引入两个局部变量：变量$color 的值为'yellow'，变量$fruit 的值为'banana'。然后调用被包含文件 include2.php，在被包含文件中，首先声明$color 是全局变量，这样，在该被包含文件中以及返回自定义函数后的后续代码中，涉及的变量$color 都是全局变量，而涉及的变量$fruit 仍然是局部变量。在函数执行完返回到主文件后，涉及的变量$color 和$fruit 则都是全局变量。

4.4.2 包含文件位置

包含文件语句中的 filespec 指定被包含的文件。如果 filespec 只指定了文件名，或者还指定了相对路径，那么 PHP 解释器会从什么位置去定位这个被包含文件呢？这里涉及 3 个位置：

（1）include_path 目录。在 php.ini 文件中定义。

（2）调用脚本文件所在目录。

（3）当前工作目录，即当前被客户请求的脚本文件所在的目录。

PHP 解释器将按上述顺序指定的位置去定位 filespec 文件，即首先到 include_path 目录去寻找被包含文件，如果找不到再到调用脚本文件所在目录去寻找，如果还找不到就到当前工作目录去寻找。

如果 filespec 指定了被包含文件的绝对路径，比如在 Windows 操作系统中，指定了盘符和路径，那么就忽略上述位置，直接读取指定位置的被包含文件。

如果 filespec 指定的文件路径以.或..开头，那么这个路径是相对于当前工作目录的。比如，./inc/demo.php 指的是当前工作目录下 inc 子目录中的 demo.php 文件，../inc/demo.php 指的是当前工作目录的父目录下 inc 子目录中的 demo.php 文件。

4.5　实例：创建管理员子系统主页

本实例初步创建教务选课系统管理员子系统的主页，如图 4-6 所示。其中管理员登录表单还有待实现。



图 4-6　管理员子系统主页

本实例结果包含 4 个文件：作为主页的 PHP Web 页文件，文件名为 index_admin.php；作为页面头模块的 PHP 文件，文件名为 header_admin.php；作为页面脚模块的 PHP 文件，文件名为 footer_admin.php；作为 logo 的图像文件，文件名为 logo.png。前 3 个文件都保存在 admin 文件夹里，文件 logo.png 保存在“源文件”结点下的 image 子文件夹里。

下面是主页文件 index_admin.php 的代码：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>首页</title>
    <link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
```

```
7.     </head>
8.     <body>
9.       <?php include("header_admin.php"); ?>
10.
11.      <div style="width: 90%; min-height: 150px; margin: 0 auto; padding: 5px;">
12.        <?php
13.        echo "管理员登录表单";
14.        // include("login_admin.php");
15.        ?>
16.      </div>
17.
18.      <?php include("footer_admin.php"); ?>
19.    </body>
20. </html>
```

其中，第 14 行被注释掉的代码的作用是在页面主区包含模块文件 login_admin.php，该文件的功能是呈现登录表单并完成对登录数据的处理。本书将在第 9 章介绍该模块文件的实现。

下面是页面头模块文件 header_admin.php 的代码：

```
<!--
 *  功能：呈现管理员子系统的页面头
 *  输入：无
 -->
<style type='text/css'>
   .ah { width: 90%; background-color: #eeeeee; margin: 0 auto}
   .ah img {float: left; margin: 2px 50px 2px 2px; border-style: none;}
   .ah span {float:left; font-size: 28px; font-weight: 700;margin-top: 20px}
</style>
<div class='ah'>
   <img src='/xk/image/logo.png' />
   <span style='color: rgb(178, 200, 187)'>选课系统</span>
   <span style='color: rgb(114, 83, 52)'>•</span>
   <span style='color: rgb(69, 137, 148)'>管理员子系统</span>
   <div style='clear: both'></div>
</div>
```

下面是页面脚模块文件 footer_admin.php 的代码：

```
<!--
 *  功能：呈现管理员子系统的页面脚
 *  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
 -->
<style type='text/css'>
```

```
    .hl {
      width: 90%; height: 0; margin:20px auto 0 auto;
      border-style: solid; border-color: #458994;border-width:1px 0 0 0
    }
    .af {width: 90%; margin: 0 auto 0 auto; }
    .af img {float: right; width: 33px; height: 22px; margin: 2px 5px 0 0}
    .af span {float:left;font-family:sans-serif,宋体; margin: 12px 0 0 5px}
</style>
<div class='hl'></div>
<div class='af'>
    <img src='/xk/image/logo.png'>
    <span>Copyright&copy; 2016  首都经济贸易大学 信息学院</span>
    <div style='clear: both'></div>
</div>
```

页面头模块文件和页面脚模块文件主要使用了各自内部样式表定义的规则。页面脚模块文件也用到了外部样式表 xk.css 中的一些基本样式。当单独运行这些模块文件时，最好能链入该外部样式表文件。

习 题 4

1. 写出下面 PHP 代码运行的输出结果。

（1）

```
$i=10;
$n = $i++;
echo $n, $i++, ++$i;
```

（2）

```
echo 'Testing ' . 1 + 2 . '45';
```

（3）

```
echo 1+'2'.'1';
```

（4）

```
$str1 = "";
$str2 = 0;
var_dump($str1==$str2);
```

（5）

```
$str1 = 0;
$str2 = '0';
var_dump($str1===$str2);
```

(6)

```
$str1 = null;
$str2 = false;
var_dump($str1!=$str2);
```

(7)

```
$a1 = null;
echo isset($a1) ? 'true' : 'false';
```

(8)

```
$a1 = false;
echo empty($a1) ? 'true' : 'false';
```

(9)

```
$a1 = '0';
echo empty($a1) ? true : false ? 'true' : 'false';
```

(10)

```
$s = 0; $a = 60;
if($a >= 50) $s = 100 + $a;
$s = 100;
echo $s;
```

(11)

```
$i=1;
if($i===0) {
   echo "zero";
} elseif($i===1) {
   echo "one";
} elseif($i===2) {
   echo "two";
} else {
   echo "default";
}
```

(12)

```
$i=1;
switch($i) {
   case 0: echo "zero";break;
   case 1: echo "one";
   case 2: echo "two";
   default: echo "default";
```

```
}
```

（13）

```
$s=0; $i=0;
while($i++<10) {
   $s+=$i;
}
echo $s;
```

（14）

```
for($i=0; $i<3; $i++) {
   for($j=10; $j<30; $j+=10) {
      echo $i+$j;
      if($i>1) continue 2;
   }
}
```

（15）

```
$a = array(1, 2, 3, 4, 5);
foreach($a as $key=>&$value) {
   $value = $value - $key;
}
print_r($a);
```

2. 简答题

（1）简述运算符“==”与“===”的区别。试举一个用“==”判断为真，但用“===”判断为假的例子。

（2）有变量$a=0、$b=1、$c=2，请写出下面表达式的类型与值：

```
$a++
$a&&$b
$a&$b
$a>=0||$b==0&&$c<2
$x=$y=$c
```

（3）简述 include 语句与 require 语句、include 语句与 include_once 语句的区别。

第 5 章　PHP 函数

本章主题：

- 函数的声明与调用；
- 函数参数；
- 函数返回值；
- 变量函数；
- 匿名函数；
- 日期和时间函数。

函数通常是指能够完成某个特定功能、可被其他程序调用的一段代码。在软件开发中，函数是功能模块化与代码重用的一项技术，有助于降低软件开发和维护的难度、提高软件开发的生产率以及改善软件质量。在 PHP 中，函数也是页面模块化和页面代码重用的一种手段。除了系统内置的函数，用户可以自定义所需的函数。

5.1　函数的声明与调用

这里先介绍函数声明和函数调用的基本格式及用法，更多的细节会在下面各节依次介绍。

5.1.1　函数声明

函数声明以关键字 function 开始，接着给出函数名。函数名后是一对括号，括号内是可选的形参名列表，各形参之间用逗号分隔。最后给出函数体，即调用该函数时要执行的代码，代码放置在一对花括号内。

```
function <function_name>([<$arg> [,<$arg>]*]) {
   // 要执行的代码;
}
```

函数名的命名规则：

（1）函数名不能和已有的函数重名；

（2）函数名称只能包含字母、数字和下画线；

（3）函数名称不能以数字开头；

（4）长度不限，对大小写不敏感。

函数名不区分大小写，但按其声明时的大小写形式来调用它，是一种好的习惯。函数名应尽量精炼且有意义，例如一个用于产生页脚的函数，用 pagefooter 或 page_footer 作为名称是不错的。

PHP 不支持函数重载，即在一个 PHP 页面文件中，不允许存在两个具有相同函数名的函数，即使他们声明有不同的形参。由于系统内置的函数可应用于任何一个 PHP 页面文件中，所以也不能声明一个与系统内置函数同名的自定义函数。

函数体内包含的代码可以是任何合法的 PHP 代码，例如赋值语句、表达式语句、流程控制语句、函数调用语句，甚至是其他函数或类的声明。

自定义函数的作用域是全局的。例如，可以在一个函数外调用声明在该函数内的函数，也可以在一个函数内调用声明在该函数外的函数。

【例 5-1】 编写一个 isLeap 函数，用于判断指定年份$year 是否为闰年。闰年是指能被 400 整除或者能被 4 整除但不能被 100 整除的年份。代码如下：

```
1.  <?php
2.  function isLeap($year) {
3.     return $year%400===0 || $year%4===0 && $year%100!==0;
4.  }
5.
6.  var_dump(isLeap(1990));
7.  ?>
```

下面是代码运行的输出结果：

```
bool(false)
```

这里，第 2~4 行是题目要求编写的函数，第 6 行代码涉及对该函数的调用。当访问一个 PHP 文件时，文件内声明的函数不会自动执行，只有在其他代码调用它时，它才会执行。

5.1.2 函数调用

函数声明应该出现在 PHP 开始标签和结束标签之间，但在所在页面加载时函数并不会立即执行。函数只有在被调用时才会执行。与调用 PHP 系统内置函数一样，可以通过函数名调用用户自定义函数，函数名后的括号内给出要传递给函数形参的值。函数调用格式如下：

```
<function_name>([<expr> [,<expr>]*]);
```

内置函数可以在所有的 PHP 脚本中使用，但对用户自己声明的函数，只能在包含它的页面内使用。这里，并不要求先声明后使用，通常可以在一个函数声明之前调用该函数。

页面中的 PHP 代码可以调用函数，函数内的 PHP 代码也可以调用其他函数，甚至是该函数本身。这种直接或间接调用自身的函数被称为递归函数。

递归的基本思想是不断把问题分解成规模较小的同类问题，直到分解形成的问题因规模足够小而能直接求得解为止。而这个过程中出现的每个问题都将由规模较小的同类问题的解而求得解。

递归思想反映到代码设计上，体现为一个函数直接或间接地调用自身。这里，函数表示问题的解法，函数参数表示问题的规模。一个递归函数的代码一般应包括两部分：一部分是如果问题的规模足够小，问题的解是什么？另一部分是如果问题的规模不是足够小，

那么它应该如何由规模较小的同类问题的解而求得解。

【例 5-2】 声明一个函数 fib，其功能是求斐波纳契数列的第 n 个数。所谓斐波纳契数列是这样一个数列：前两个数都是 1，第三个数是前两个数之和，以后的每个数都是其前两个数之和。代码如下：

```
1.  <?php
2.  echo fib(20);
3.
4.  function fib($n) {
5.     if($n===1 || $n===2) return 1;
6.     return fib($n-1) + fib($n-2);
7.  }
8.  ?>
```

下面是代码运行的输出结果：

```
6765
```

这里，第 4~7 行是题目要求编写的函数，第 2 行代码包含对该函数的调用。该函数采用递归方法实现，函数体分为两部分。第 5 行代码是第一部分：如果问题规模较小，即变量$n 的值为 1 或 2，则问题的解为 1；第 6 行代码是第二部分：将问题分解为两个规模较小的子问题，它们的规模分别为$n-1 和$n-2，并将两个子问题的解的和作为整个问题的解。

一般来说，使用递归的程序代码会更简洁，也更容易理解，但递归代码的执行效率较低，所以应尽量避免使用。

5.2 函 数 参 数

PHP 支持按值传递参数和按引用传递参数两种方式，也具有默认参数值以及可变长参数等特性。

5.2.1 形参与实参

有些函数可能需要从外部接收一些数据，然后再进行数据处理并完成相应的功能，这时可以为函数定义相应的参数，通常称之为形参。形参被定义在函数名之后，括号内部。一个函数的形参数目不限，两个形参之间用逗号分隔。函数形参类似于在函数内定义的局部变量，在函数内有效。

当调用包含形参的函数时，应提供相应的参数值，通常称之为实参。实参是表达式，两个表达式之间用逗号分隔，各表达式从左到右进行计算，计算结果被传递给对应的形参。参数类型既可以是标量类型，也可以是复合类型。通常，实参的数目和类型应与函数形参的数目和所需的类型相一致。如果实参的数目少于形参的数目，系统将给出警告（Warning）信息，未获得值的形参将是未定义的。如果实参的数目多于形参的数目，则多余的实参被忽略。

默认情况下，函数的参数是按值传递的。这意味着，即使实参是变量，当函数对形参

的值进行改变后，也不会影响函数外部实参的取值。

有些情况下，会希望函数对形参的值的改变能同步反映到函数外部的实参变量上。按引用传递可以满足这一要求。所谓按引用传递是指，当实参是变量时，传递的将是变量的引用而非变量的值，其结果是形参和实参变量引用同一个内存单元、代表同一个变量。要实现按引用传递，只需在形参名前加上符号“&”，例如&$arg。

【例 5-3】 按值传递和按引用传递。代码如下：

```
<?php
function pass_arg($a, &$b) {
   $a .= ' and something extra.';
   $b .= ' and something extra.';
}
$x = "books";
$y = "bags";
pass_arg($x, $y);
echo $x."<br />";
echo $y;
?>
```

下面是代码运行的输出结果：

```
books
bags and something extra.
```

在上述函数 pass_arg 中，第 1 个参数按值传递，所以实参$x 和形参$a 是两个独立的变量。第 2 个参数按引用传递，所以实参$y 和形参$b 引用同一个内存单元。当改变形参$b 的值时，实参$y 的值也随之改变。

要实现按引用传递，需要：

（1）形参名前加&；

（2）实参是变量。

5.2.2　参数的默认值

在声明函数时，可以为函数形参指定默认值。形参的默认值必须是常量表达式，通过运算符“=”给其赋值。例如，下面的函数声明：

```
function <function_name>(<$arg1>,<$arg2>=<constant_expression>) { … }
```

其中，形参$arg2 指定了一个默认值。

当调用函数时，如果没有为具有默认值的形参传递值，那么默认值就会自动赋给该形参。例如，在下面的函数调用中，表达式 expr1 的值传递给形参$arg1，而形参$arg2 将取默认值。

```
<function_name>(<expr1>);
```

具有默认值的形参可以有多个，但必须放置在其他形参后面。当调用函数时，对不具

有默认值的形参应该指定相应的实参，对具有默认值的形参，可以指定实参，也可以没有。如果只指定了部分实参，那么各参数将会按照从左到右的顺序进行赋值。也就是说，不能试图给后面的形参传递值，而让前面的形参取默认值。

【例 5-4】 默认参数值。代码如下：

```
<?php
function default_arg($price, $tax = 0.0675) {
  $total = $price + $price * $tax;
  return $total;
}

echo "Total cost: ".default_arg(12.5)."<br />";
echo "Total cost: ".default_arg(12.5, 0.065);
?>
```

下面是代码运行的输出结果：

```
Total cost: 13.34375
Total cost: 13.3125
```

第 1 次调用函数时，只传递了 1 个参数值 12.5。该参数值赋给形参$price，形参$tax 取默认值 0.0675。第 2 次调用函数时，传递了 2 个参数值。第 1 个参数值 12.5 赋给形参$price，第 2 个参数值 0.065 赋给形参$tax。

【例 5-5】 利用函数实现本书 3.4 节介绍的“动态水平导航栏”模块。

考虑调用外部样式表文件/xk/css/xk.css 的方便性，该例在教务选课系统项目 xk 中实现。文件名为 fnavigation_admin.php，保存在项目中 admin 子文件夹下。下面是该文件的代码：

```
<?php
/*
 *  功能：根据应用状态动态呈现导航栏
 *  输入：链入外部样式表<link rel='stylesheet' type='text/css' href='/xk/css/xk.css'/>
 *        $name：登录管理员的姓名
 *        $choice：当前页面对应的菜单项序号，取 1、2 或 3
 */
function navigation_admin($name, $choice=1) {
  echo "<div class='nav'>";
  echo "<a class='right' href='exit.php'>退出</a>";
  echo "<span class='right'>".$name."，您好"."</span>";
  echo "<a class='left ".($choice===1 ? "current" : "").
                      "' href='teacher_p.php'>浏览教师信息</a>";
  echo "<a class='left ".($choice===2 ? "current" : "").
                      "' href='course_p.php'>添加课程</a>";
  echo "<a class='left ".($choice===3 ? "current" : "").
                      "' href='opencourse_p.php'>维护开课信息</a>";
```

```
18.    echo "<div style='clear: both'></div>";
19.    echo "</div>";
20. }
21. ?>
22.
23. <link rel='stylesheet' type='text/css' href='/xk/css/xk.css'/>
24. <?php
25. $user = "刘绍军";
26. navigation_admin($user, 2);
27. ?>
```

第 2~20 行是实现动态水平导航栏的函数，其中第 2~7 行是对该函数的注释说明。第 23 ~26 行演示了对函数的调用。

与用 PHP 文件实现页面模块相比，用 PHP 函数实现页面模块时，调用者与函数之间可以通过参数传递数据，而不需要事先在调用文件处为特定变量设置值，所以更加方便，调用者和被调用者之间的耦合度更低，便于开发和维护。

PHP 文件既可以包含 PHP 代码，也可以包含 HTML 等静态代码，而 PHP 函数内只能是 PHP 代码。如果一个页面模块包含太多的 HTML 等静态代码，那么用函数来实现就会增加一些工作量，且会降低代码的可读性。

5.2.3 可变长参数

调用函数时，有时候也可能需要向函数传递数量不等的参数值。PHP 支持可变长参数。在 PHP 5.6 及以后的版本中，可变长参数通过在形参名前加符号“…”实现。可变长形参必须是形参表中最后一个形参，否则会产生一个致命错误（Fatal error）信息。可变长形参可以接收零个或多个实参值，此时这些参数值将被组织成一个数组赋给该形参。

【例 5-6】 可变长形参的使用。代码如下：

```
<?php
function sum(&$sum, ...$numbers) {
   foreach($numbers as $n) {
      $sum += $n;
   }
}

$original = 100;
sum($original, 1, 2, 3, 4, 5);
echo "sum=".$original."<br />";
sum($original, 10);
echo "sum=".$original."<br />";
sum($original);
echo "sum=".$original;
?>
```

下面是代码运行的输出结果：

```
sum=115
sum=125
sum=125
```

函数 sum 有两个形参，第 1 个形参按引用传递，第 2 个形参是一个可变长形参。例子共 3 次调用函数，第 1 次调用时，传递给形参$numbers 的是一个包含 5 个元素的数组；第 2 次调用时，传递给形参$numbers 的是一个包含 1 个元素的数组；第 3 次调用时，传递给形参$numbers 的是一个空数组。

反过来，你也可以在实参前加符号“...”，称为可变长实参。可变长实参必须是实参表中的最后一个实参，否则会产生一个致命错误（Fatal error）信息。可变长实参的类型应该是数组。这样，当传递参数时，实参数组中的各元素将被自动取出并一一赋给对应的形参。

如果可变长实参数组的元素过少，不能给相关的形参赋值，系统将给出警告（Warning）信息，未获得值的形参将是未定义的。

【例 5-7】 可变长实参的使用。代码如下：

```php
<?php
function add($a, $b, $c) {
   return $a + $b +$c;
}

$a = array(1, 2, 3);
echo add(...$a)."<br />";
echo add(10, ...$a)."<br />";
?>
```

下面是代码运行的输出结果：

```
6
13
```

例子共两次调用函数，第 1 次调用时，可变长实参数组$a 中的 3 个元素值 1、2 和 3 分别传递给了形参$a、$b 和$c。第 2 次调用函数时，第 1 个实参 10 传递给形参$a，可变长实参数组$a 中的前两个元素值 1 和 2 分别传递给了形参$b 和$c。

5.3 函数返回值

return 语句可以出现在函数中。当代码执行 return 语句时，控制将从函数中退出并返回到调用者处。

如果需要函数有一个返回值，可以使用带表达式的 return 语句。表达式可以是任意类型的，所以一个函数可以返回数值、字符串等标量类型的值，也可以返回数组、对象等复合类型的值。

【例 5-8】 数组作为参数值和返回值。代码如下：

```php
<?php
function getUser($score) {
   $user = array("20090701011", "LiMing", $score[0], $score[1], $score[2]);
   return $user;
}

$score = array(90, 82, 92);
$a = getUser($score);
print_r($a);
?>
```

下面是代码运行的输出结果：

```
Array ( [0] => 20090701011 [1] => LiMing [2] => 90 [3] => 82 [4] => 92 )
```

调用函数时，传递的参数值是一个数组，包含某个学生的三门课的成绩。函数创建了一个新的数组，除了包含成绩，还包含学生的学号和姓名。函数返回新创建的数组。

一个函数中可以包含多个 return 语句，代码一旦执行到 return 语句就从函数中返回。一般情况下，出现在函数体末尾的不带表达式的 return 语句可以被省略。当省略 return 语句或 return 语句不带表达式时，函数返回 NULL 值。

函数可以返回一个值，也可以返回一个引用。要让函数返回一个引用，需要：

（1）在函数声明时，函数名前使用&；

（2）return 语句所带的表达式是变量；

（3）调用函数时，函数名前使用&。

【例 5-9】 函数返回引用。代码如下：

```php
<?php
function &myFunc() {
   static $x = 0;
   $x++;
   echo '$x='.$x."<br />";
   return $x;
}

$y = &myFunc();
$y = 10;
myFunc();
?>
```

下面是代码运行的输出结果：

```
$x=1
$x=11
```

这里，$x 是一个静态变量。第 1 次调用函数时，静态变量$x 初始化为 0，接着加 1 后输出。函数返回$x 的引用并赋给变量$y，这样变量$x 和$y 就引用相同的内存单元。当把$y

置为 10 后，变量$x 的值也为 10。第 2 次调用函数时，静态变量$x 不再初始化，加 1 后变为 11，然后输出。

5.4 变量函数

PHP 支持变量函数的概念。这意味着，如果一个变量名后紧跟着括号，PHP 将试着调用与该变量值同名的函数。

【例 5-10】 可变函数。代码如下：

```
1.  <?php
2.  function foo() {
3.     echo "In foo()<br />";
4.  }
5.  function bar($p = "default") {
6.     echo "In bar($p)<br />";
7.  }
8.  function func($func_name) {
9.     $func_name();
10. }
11.
12. func("foo");
13. func("bar");
14. ?>
```

下面是代码运行的输出结果：

```
In foo()
In bar(default)
```

这里，第 9 行代码是一个变量函数调用，根据变量$func_name 的不同取值，调用不同的函数。

变量函数中，变量指定的函数名既可以是用户自定义函数名，也可以是系统内置函数名，但不能是语言结构名。变量函数不适用于语言结构，例如 echo、print、unset、isset、empty、include 和 require 等。

下面代码中，第 4 行代码是合法的，因为 is_null 是一个内置函数；第 6 行代码是不合法的，因为 isset 是一个语言结构。

```
1.  <?php
2.  $var = true;
3.  $a = "is_null";
4.  var_dump($a($var));
5.  $b = "isset";
6.  var_dump($b($var));
7.  ?>
```

如果不存在与变量值同名的自定义函数或系统内置函数，变量函数调用时将产生一个致命错误（Fatal error）信息。为了防止这类错误，可以在调用函数之前使用 PHP 的系统内置函数 function_exists()来判断该函数是否存在。例如：

```
function func($func_name) {
   if(function_exists($func_name)){
      $func_name();
   }
}
```

函数 function_exists()返回布尔型。如果指定的参数值是某个自定义函数或系统内置函数的名称，则函数返回 true；否则，函数返回 false。

5.5 匿 名 函 数

匿名函数是指没有指定名称的函数。在定义匿名函数时，关键字 function 后面直接跟括号和形参表。下面介绍匿名函数的两种用法。

5.5.1 匿名函数作为变量值

可以把匿名函数作为一个表达式赋给一个变量，然后就可以通过该变量来调用匿名函数了。把一个匿名函数赋值给一个变量的语法与普通变量赋值的语法没有区别，最后也要加上分号。

在内部处理中，PHP 系统会自动把匿名函数转换成内置类 Closure 的一个实例对象，然后再把该实例对象赋给变量。

【例 5-11】 匿名函数作为变量值。代码如下：

```
<?php
// 给变量$green 赋一个匿名函数
$greet = function($name) {
   echo "Hello ", $name, "<br />";
};                  // 函数末尾需要分号

$greet('World');    // 通过变量调用函数
$greet('PHP');
?>
```

下面是代码运行的输出结果：

```
Hello World
Hello PHP
```

使用 use 语言结构，匿名函数可以使用父作用域中的变量。其格式如下：

```
<变量>=function(<$arg>) use(<$var>) {…};
```

这里，$var 应该是匿名函数的父作用域中存在的一个变量。所谓匿名函数的父作用域是指匿名函数声明处的作用域，或者说，只要是在匿名函数声明处可以使用的变量，就可以通过 use 结构的声明，使其在匿名函数内可用。另外这一使用发生在处理匿名函数声明时，而不是在调用匿名函数时。下面通过例子说明这一使用过程。

【例 5-12】 使用父作用域中变量的值。代码如下：

```
1.  <?php
2.  $hi = "good morning";
3.  $greet = function($name) use($hi) {
4.     echo "Hello ", $name, ", ", $hi, "<br />";
5.  };
6.
7.  $greet('LiMing');
8.  $hi = "good afternoon";
9.  $greet('LiMing');
10. ?>
```

下面是代码运行的输出结果：

```
Hello LiMing, good morning
Hello LiMing, good morning
```

这里，匿名函数声明处（第 3 行）可以使用的变量是在第 2 行处声明的全局变量$hi，而通过 use($hi)的声明，匿名函数内就可以使用全局变量$hi 的值了。需要注意的是，use 中指定的变量$hi 是仅在匿名函数内可用的局部变量，而非全局变量$hi。它们名字相同，却是两个不同的变量。当 PHP 处理匿名函数声明时，就会将全局变量$hi 的值传递给同名的局部变量$hi。但在函数调用时，这一传递过程是不会发生的。

在上述代码运行时，尽管第 2 次调用函数前，全局变量$hi 的值改变了，但调用函数输出的结果与第 1 次调用函数输出的结果完全一样。这就是因为匿名函数内使用的局部变量$hi 的值在系统处理该匿名函数声明时就已经确定了。之后，不管何时调用匿名函数，函数内的局部变量$hi 的值是不会改变的。

也可以采用按引用传递的方式，将父作用域中的变量的引用传递给匿名函数内使用的局部变量，使两个变量引用相同的内存单元、始终表示相同的值。例如，如果把上述匿名函数声明中的 use($hi)改为 use(&$hi)，那么输出结果将是：

```
Hello LiMing, good morning
Hello LiMing, good afternoon
```

5.5.2 用作回调类型参数的值

函数的形参可以接收标量类型的数据，也可以接收复合类型的数据，甚至可以接收一个函数。例如下面代码：

```
<?php
```

```
function myFunc($name, $func) {
   $func($name);
}
function display($str) {
   echo "Hello ", $str, "<br />";
}
myFunc('World',"display");
?>
```

函数 myFunc 有两个形参，其中第 2 个形参$func 接收一个函数名。这里，用于接收函数的形参被称为回调类型（callable）参数，传递给这种参数的函数（这里为函数 display）被称为回调函数。回调函数并未在主程序中被调用，而是由函数 myFunc 在一定的时机自主调用。

这段代码也可以采用匿名函数的方式，即不事先独立声明函数 display，而是在调用函数 myFunc 时，声明一个匿名函数为参数$func 赋值，如例 5-13 所示。

【例 5-13】 匿名函数用作回调类型参数的值。代码如下：

```
<?php
function myFunc($name, $func) {
   $func($name);
}

myFunc('World', function($str) {
        echo "Hello ", $str, "<br />";
    }
);
?>
```

下面是代码运行的输出结果：

```
Hello World
```

5.6　日期时间函数

本节介绍几个常用的系统内置日期时间函数。

1. time 函数

time 函数可以获得当前时间戳，语法格式如下：

```
int time()
```

函数返回从 UNIX 纪元（格林尼治时间 1970 年 1 月 1 日 00:00:00）到当前时间经过的秒数，即当前的 UNIX 时间戳。

2. mktime 函数

mktime 函数可以根据日期时间信息获得相应的时间戳，语法格式如下：

```
int mktime([int $hour [, int $minute [, int $second [, int $month [, int $day
[, int $year]]]]]])
```

按时、分、秒、月、日、年的顺序指定各参数，各参数指定的日期时间信息是基于默认时区的。函数返回与指定日期时间信息相应的时间戳，即从 UNIX 纪元到指定时间经过的秒数。

如果缺省某些参数，则其值取当前日期时间的相应数据。

3. date 函数

date 函数可以将时间戳转化为直观的日期时间字符串，语法格式如下：

```
string date(string $format [, int $timestamp])
```

函数按照指定的格式串$format 对指定的时间戳$timestamp 进行格式化，返回格式化产生的、包含有基于默认时区的日期时间信息的字符串。如果缺省$timestamp，就使用当前的 UNIX 时间戳，即 time()的返回值。

格式串由格式符和普通文本组成。格式符描述了需要包含的相应的日期时间文本，普通文本则会原样保留在返回的字符串。表 5-1 给出了常用的格式符。

表 5-1　date 函数支持的常用格式符

格式符	说　　明	返回值例子
D	月份中的第几天，有前导零的 2 位数字	01 ~31
J	月份中的第几天，没有前导零	1 ~ 31
D	星期几（3 个字母的缩写）	Mon ~ Sun
l	星期几（完整的英文单词）	Sunday ~ Saturday
N	星期几（ISO-8601 格式数字）	1（表示星期一）~7（表示星期天）
w	星期几（数字）	0（表示星期天）~6（表示星期六）
z	年份中的第几天	0 ~ 365
M	月份（三个字母的缩写）	Jan ~ Dec
F	月份（完整的英文单词）	January ~ December
m	月份（有前导零的两位数字）	01 ~ 12
n	月份（没有前导零）	1 ~ 12
t	所在月份所应有的天数	28 ~ 31
L	是否为闰年	如果是闰年为 1，否则为 0
Y	4 位数字表示的完整年份	例如：1999 或 2003
y	2 位数字表示的年份	例如：99 或 03
a	小写的上午和下午值	am 或 pm
A	大写的上午和下午值	AM 或 PM
g	小时，12 小时格式，没有前导零	1 ~ 12
G	小时，24 小时格式，没有前导零	0 ~ 23
h	小时，12 小时格式，有前导零	01 ~ 12

续表

格式符	说　　明	返回值例子
H	小时，24 小时格式，有前导零	00 ~ 23
i	分钟数，有前导零	00 ~ 59
s	秒数，有前导零	00 ~ 59
e	时区标识	例如：UTC，GMT，Atlantic/Azores

4. getdate 函数

getdate 函数可以根据时间戳获得相应的日期时间信息，语法格式如下：

```
array getdate([int $timestamp = time()])
```

函数返回一个根据指定时间戳$timestamp 得出的、包含有基于默认时区的日期时间信息的关联数组。如果缺省$timestamp，就使用当前的 UNIX 时间戳，即 time()的返回值。返回的关联数组中各元素的键与值的说明如表 5-2 所示。

表 5-2　getdate 函数返回的关联数组中元素的键与值

键名	说　　明	返回值例子
"seconds"	秒的数字表示	0 ~ 59
"minutes"	分钟的数字表示	0 ~ 59
"hours"	小时的数字表示	0 ~ 23
"mday"	月份中第几天的数字表示	1 ~ 31
"wday"	星期中第几天的数字表示	0 (周日) ~ 6 (周六)
"mon"	月份的数字表示	1 ~ 12
"year"	4 位数字表示的完整年份	例如： 1999 或 2003
"yday"	一年中第几天的数字表示	0 ~ 365
"weekday"	星期几的完整文本表示	Sunday ~ Saturday
"month"	月份的完整文本表示	January ~ December

5. date_default_timezone_set 函数

很多日期时间函数都是基于默认时区的。例如对于同一个时间戳，在不同的时区有不同日期时间信息。date_default_timezone_set 函数可以设置默认时区，其语法格式如下：

```
bool date_default_timezone_set(string $timezone_identifier)
```

函数为当前脚本中所有日期时间函数设置一个默认时区。参数$timezone_identifier 指定时区标识符，如果参数值有效，函数返回 true，否则返回 false。中国大陆的时区标识符可用 PRC。

除了用此函数，还可以在 php.ini 文件中通过设置 date.timezone 来设置默认时区。这样设置的默认时区对所有脚本都是有效的。

date_default_timezone_get 函数可以返回当前脚本中所有日期时间函数所使用的默认时区。其语法格式如下：

```
string date_default_timezone_get(void)
```

【例 5-14】 日期时间函数的使用。代码如下：

```
<?php
date_default_timezone_set("PRC");
$time1 = mktime(15, 10, 28, 1, 20, 2016);//根据日期时间信息获取时间戳$time1
$dt = date("Y-m-d H:i:s", $time1);      //根据时间戳获得日期时间信息的字符串
echo $dt, "<br />";
$time2 = mktime(9, 50, 38, 2, 12, 2016);//根据日期时间信息获取时间戳$time2
$wday = date("w", $time2);              // 获得指定时间戳属于星期几
echo $wday, "<br />";
$lastday = date("t", $time2);           // 获得指定时间戳所在月份的天数
echo $days, "<br />";
$days = (time2 - $time1)/(24 * 60 * 60);    // 计算两个指定时间相差的天数
echo (int)$days;
?>
```

下面是代码运行的输出结果：

```
2016-01-20 15:10:28
5
29
22
```

【例 5-15】 生成日历，效果如图 5-1 所示。初始请求（不带请求参数）时，页面呈现当年、当月、当日的状态，当前日期以灰色背景显示。之后，用户可以选择年份、月份，了解所选年份月份的日期状态。



图 5-1 日历页面效果示意图

该例由 PHP 文件实现，文件名为 calendar.php，其代码如下：

```
<style type='text/css'>
   a {color: steelblue; text-decoration: none}
   .outer {width: 248px; border: 1px solid steelblue; margin: 10px auto 0 auto}
   .inter {height:30px;background-color:#eeeeee;margin: 1px 1px 5px 1px}
```

```
    td {width: 26px; padding-top: 2px; padding-right: 8px; text-align:
    right}
</style>
<?php
$year = @$_GET['year'];      // 获取请求的年份
$month = @$_GET['month'];   // 获取请求的月份
// 确定年份和月份。如果没有请求参数，年份和月份分别取当前年份和月份
// 如果是当前年份和月份，$flag 设置为 true；否则设置为 false
if(empty($year) || empty($month)) {
    $flag = true;
    $year = date("Y");
    $month = date("n");
    $day = date("j");
} else {
    if($year === date("Y") && $month === date("n")) {
        $flag = true;
        $day = date("j");
    } else {
        $flag = false;
    }
}
// days：所选月份共多少天
// $day_w_first：所选月份 1 号是星期几；$day_w_last：所选月份最后一天是星期几
$days = date("t", mktime(0, 0, 0, $month, 1, $year));
$day_w_first = date("w", mktime(0, 0, 0, $month, 1, $year));
$day_w_last = date("w", mktime(0, 0, 0, $month, $days, $year));
// 将所选月份的所有日期按顺序填入数组。若 1 号不是周日，前面填适当数量的空串，
// 若最后一天不是周六，后面填适当数量的空串。数组长度是 7 的倍数
$output_arr= array();
for($i=0; $i<$day_w_first;$i++) {
    array_push($output_arr, "");
}
for($i=1; $i<=$days; $i++) {
    if($flag && $day==$i) {
        array_push($output_arr, "<span style='background-color: gray'> $I
        </span>");
    } else {
        array_push($output_arr, "$i");
    }
}
for($i=$day_w_last+1; $i<=6; $i++) {
    array_push($output_arr, "");
}
// 为年份和月份超链接的请求参数计算参数值
$year1 = $year<=1970 ? 1970 : $year-1;
```

```
$year2 = $year>=2037 ? 2037 : $year+1;
$month1 = $month<=1 ? 1 : $month-1;
$month2 = $month>=12 ? 12 : $month+1;
?>
<!-- 呈现日历 -->
<div class="outer">
    <!-- 呈现年份和月份及相应的超链接 -->
    <div class="inter">
        <span style="float: left; margin-top: 6px">
        <?php
            echo "<a href='calendar.php?year=$year1&month=$month'>&lt; </a>";
            echo $year."年";
            echo "<a href='calendar.php?year=$year2&month=$month'>&gt;</a>";
        ?>
        </span>
        <span style="float: right; margin-top: 6px">
        <?php
            echo "<a href='calendar.php?month=$month1&year=$year'>&lt; </a>";
            echo $month."月";
            echo "<a href='calendar.php?month=$month2&year=$year'>&gt;</a>";
        ?>
        </span>
    </div>
    <!-- 呈现所选年份月份的日期状态 -->
    <table style="border-collapse:collapse">
        <tr>
        <td>日</td><td>一</td><td>二</td><td>三</td><td>四</td><td>五</td><td>六</td>
        </tr>
        <?php
        echo "<tr>";
        for($i=0; $i<count($output_arr);$i++) {
            echo "<td>$output_arr[$i]</td>";
            if(($i+1)%7===0) {
                if($i+1===count($output_arr)) {
                    echo "</tr>";              // 结束
                } else {
                    echo "</tr><tr>";          // 继续下一行
                }
            }
        }
        ?>
    </table>
</div>
```

年份和月份左右各有两个超链接，每个超链接都携带了两个请求参数：year 和 month。页面中用下面代码获取这两个请求参数的值：

```
$year = @$_GET['year'];
$month = @$_GET['month'];
```

其中，$_GET 是超全局变量，是一个包含所有 GET 请求参数的数组。首次请求时，由于没有携带查询参数，$_GET 数组不存在相应的元素，访问时将会出现 Notice 错误信息，这里用运算符@抑制了这个错误信息。

习 题 5

1．写出下面 PHP 代码运行的输出结果

（1）

```
function m($val){
   ++$val;
}
$val = 10;
m($val);
echo $val;
```

（2）

```
function m(&$val){
   ++$val;
}
$val = 10;
m($val);
echo $val;
```

（3）

```
function get_arr($arr){
   unset($arr[0]);
}
$arr1 = array(1, 2);
get_arr($arr1);
echo count($arr1);  // count($arr1)返回数组大小
```

（4）

```
function get_arr(&$arr){
   unset($arr[0]);
}
$arr1 = array(1, 2);
get_arr($arr1);
```

```
echo count($arr1);  // count($arr1)返回数组大小
```

(5)

```
function myFunc($a, $b = 10, $c = 100) {
   return $a + $b + $c;
}
echo myFunc(10, 50);
```

(6)

```
function sum(...$numbers) {
   $acc = 0;
   foreach ($numbers as $n) {
      $acc += $n;
   }
   return $acc;
}
echo sum();
```

(7)

```
function func($arr) {
   $arr1 = array();
   foreach($arr as $key=>$value) {
      $arr1[$key] = $key.$value;
   }
   return $arr1;
}
$arr = array(1, 2, 3, 4, 5);
$arr = func($arr);
echo $arr[1];
```

(8)

```
function &myFunc($a) {
   $GLOBALS['x'] += $a;
   return $GLOBALS['x'];
}
$x = 100;
$y = &myFunc(50);
echo $x."-".$y;
```

(9)

```
function func($a) {
   return $a**2;
}
```

```
$var = "func";
echo $var(3)+$var(4);
```

（10）

```
function f1($n) {
   return $n+1;
}
function f2($n) {
   return $n+10;
}
$f = function($n, $a) {
   $n = $a($n);
   return $n;
};
echo $f($f(10, "f1"), "f2");
```

2. 根据要求写 PHP 代码

（1）声明一个函数 gcd($m, $n)，其功能是计算并返回两个整数$m 和$n 的最大公约数。

（2）声明一个函数 daysInMonth，函数接收两个参数$y 和$m，其功能是计算并返回指定年份（$y）中指定月份（$m）包含的天数。参数$y 的默认值为 2000。

（3）按类似“2016-01-30 11:07:54 AM”的格式要求，输出当前时间在当前默认时区的日期时间信息。

（4）按类似“2016-01-30 星期 6”的格式要求，输出当前时间在 PRC 时区的日期时间信息。

（5）将 PRC 时区的时间“2012 年 12 月 10 日 15 点 30 分 20 秒”转换为一个表示时间戳的整数，并赋给变量$t。

3. 简答题

（1）如何定义一个函数？函数名区分大小写吗？

（2）PHP 中函数传递参数的方式有哪两种？两者有什么区别？

（3）什么是匿名函数？举例说明匿名函数的用法。

第 6 章　处理字符串

本章主题：

- 长度与去空；
- 大小写转换与比较；
- 子串处理；
- 分割和连接字符串；
- 格式化输出；
- 字符串特殊处理；
- 正则表达式；
- PHP 模式匹配函数。

在实际应用中，字符串处理是必不可少的，有些时候甚至是非常关键的。前面有关章节介绍了字符串的表示、连接和比较运算以及字符串和其他类型数据之间的转换等知识。本章将讨论如何使用 PHP 内置函数来处理字符串，包括字符串的长度与去空、大小写转换与比较、子串的查找与替换、字符串的分割与连接以及格式化输出等内容。

正则表达式（Regular Expression）是一种计算机语言，用于描述字符串的模式。它为解决与字符串文本处理相关的许多常见任务提供了有效而简捷的方法。比如要验证一个作为身份证的字符串是否有效：共有 18 位，前 17 位是数字，最后一位可以是数字也可以是字母 X 或 Y。那么，可以先写出相应的正则表达式：'/^[0-9]{17}([0-9]|X|Y)$/'，然后用它去匹配目标字符串。若能匹配成功，则表明目标字符串是有效的。除此之外，还可以利用正则表达式进行子串的匹配查找与替换等操作。本章后面两节分别介绍正则表达式以及 PHP 中使用正则表达式的相关函数。

6.1　长度与去空

字符串的长度是指字符串包含的字节个数或字符个数。字符串去空通常是指去除字符串首端或尾端的空白字符。

6.1.1　字符串长度

字符串的长度包括字节长度和字符长度。

1. 字节长度

strlen 函数可以获取字符串的字节长度，其语法格式如下：

```
int strlen(string $str)
```

函数返回指定字符串$str 的字节长度，即字符串占用的字节个数（一个字符可能占用多

个字节）。若$str 为空串或 NULL，函数返回 0。

2. 字符长度

mb_strlen 函数可以获取字符串的字符长度，其语法格式如下：

```
mixed mb_strlen(string $str [, string $encoding])
```

函数返回字符串$str 的字符长度，即字符串包含的字符个数。由于在不同的字符集中一个字符会有不同的编码和所需字节数，所以计算字符串的字符长度时，需要指定字符串所采用的字符集。参数$encoding 指定字符串采用的字符集，通常就是当前脚本文件所采用的字符集。

若$str 为空串或 NULL，函数返回 0。如果$encoding 指定的不是一个字符集的名称，会产生一个警告（Warning）信息，函数返回 false。如果$encoding 指定的字符集与字符串$str 实际采用的字符集不一致，函数返回的结果可能是不正确的。

6.1.2 字符串去空

去空函数包括 trim、ltrim 和 rtrim，它们的格式和功能如下：

格式：

```
string trim($str [, $charlist])
```

功能： 去除首尾两端空白字符或指定字符。

格式：

```
string ltrim($str [, $charlist])
```

功能： 去除首端空白字符或指定字符。

格式：

```
string rtrim($str [, $charlist])
```

功能： 去除尾端空白字符或指定字符。

函数从指定字符串$str 的首端和（或）尾端去除空白字符或指定字符，然后返回去空后的字符串。

默认情况下（缺省参数$charlist），函数去除空白字符。这里空白字符包括：

- 空格（\x20）；
- 回车符（\r）；
- 换行符（\n）；
- 水平制表符（\t）；
- 垂直制表符（\x0B）；
- 空字符（\0）。

函数也可以从字符串首尾两端去除某些特定的字符。这些字符由参数$charlist 指定。该参数在指定需要去除的字符时，可以使用“..”（两点）指定一个字符范围，如\x61..\x7a，表示所有的小写字母。

【例 6-1】 字符串的长度与去空。代码如下：

```
1.  <?php
2.  header("Content-type:text/html;charset=UTF-8");
3.  $str = "Java 语言";
4.  echo strlen($str)."<br/>";                              // 输出：10
5.  echo mb_strlen($str, "UTF-8")."<br/>";                  // 输出：6
6.  $str1 = " 1234567\r\n";
7.  echo strlen($str1)."-".strlen(trim($str1))."<br/>";// 输出：10-7
8.  $str2 = "12abc345xyz67";
9.  echo trim($str2, "\x30..\x39");                         // 输出：abc345xyz
10. ?>
```

这里假设脚本文件采用的字符集为 UTF-8。第 2 行代码是 PHP 的内置函数 header，其功能是产生一个 HTTP 响应域，告诉浏览器：响应内容是一个 HTML 文件，采用 UTF-8 字符集。

6.2 大小写转换与比较

这里介绍有关字符串大小写转换和字符串大小比较的函数。

6.2.1 大小写转换

可以将字符串中的小写字母转换成大写字母，也可以将字符串中的大写字母转换成小写字母。

1. 小写转大写

strtoupper 函数实现小写转大写的功能，其语法格式如下：

```
string strtoupper(string $string)
```

将参数字符串$string 中所有小写字母转换为大写字母产生一个新的字符串并返回。

2. 大写转小写

strtolower 函数实现大写转小写的功能，其语法格式如下：

```
string strtolower(string $string)
```

将参数字符串$string 中所有大写字母转换为小写字母产生一个新的字符串并返回。

6.2.2 字符串比较

在 PHP 中，字符串既可以按字典顺序比较，也能够按自然顺序比较。

1. 按字典顺序比较

所谓按字典顺序比较两个字符串是指从左到右比较两个字符串对应位置上的字符，如果发现两个字符不同，就按这两个字符在字典中的先（小）后（大）次序决定两个字符串的大小。

按字典顺序比较两个字符串的函数包括 strcmp 和 strcasecmp，其语法格式如下：

```
int strcmp(string $str1, string $str2)
int strcasecmp(string $str1, string $str2)
```

说明：

（1）在$str1 大于、等于和小于$str2 的情况下分别返回大于 0、等于 0 和小于 0 的整数。

（2）函数 strcmp 比较时区分字母大小写，函数 strcasecmp 比较时不区分大小写。

也可以利用关系运算符来比较字符串的大小，它们也是按字典顺序来比较的，但比较的结果类型是布尔型的。

2. 按自然顺序比较

按自然顺序比较两个字符串的函数包括 strnatcmp 和 strnatcasecmp，其语法格式如下：

```
int strnatcmp(string $str1, string $str2)
int strnatcasecmp(string $str1, string $str2)
```

说明：

（1）在$str1 大于、等于和小于$str2 的情况下分别返回大于 0、等于 0 和小于 0 的整数。

（2）自然顺序适合于包含数字的项目名称的排序，如有以下字符串（省略了定界符）：Fig.1、Fig.2、Fig.3、…、Fig.9、Fig.10、Fig.11、Fig.12。

若按字典顺序比较，其从小到大的排列顺序是 Fig.1、Fig.10、Fig.11、Fig.12、Fig.2、Fig.3、…、Fig.9。

若按自然顺序比较，其从小到大的排列顺序则是 Fig.1、Fig.2、Fig.3、…、Fig.9、Fig.10、Fig.11、Fig.12。

（3）函数 strnatcmp 比较时区分字母大小写，函数 strnatcasecmp 比较时不区分大小写。

说明：有关自然顺序的知识可以在 http://www.naturalordersort.org/网站上获得进一步了解。

【例 6-2】 字符串比较。代码如下：

```
1.  <?php
2.  $arr1 = $arr2 = array("img12.png", "img10.png", "img2.png", "img1.png");
3.  usort($arr1, "strcmp");
4.  print_r($arr1);
5.  echo "<br/>";
6.  usort($arr2, "strnatcmp");
7.  print_r($arr2);
8.  ?>
```

下面是代码运行的输出结果：

```
Array ( [0] => img1.png [1] => img10.png [2] => img12.png [3] => img2.png )
Array ( [0] => img1.png [1] => img2.png [2] => img10.png [3] => img12.png )
```

其中第 3 行和第 6 行代码都用到了 PHP 内置的 usort 函数。该函数可以对指定数组的各

元素依据指定的比较函数进行比较排序。详细内容见第 10 章。

6.3 子 串 处 理

子串是指字符串内连续的一部分字符，包括空串和字符串本身。子串处理涉及子串的获取、子串的查找、子串的替换等操作。

6.3.1 访问单个字符

可以通过方括号或花括号访问字符串中的某单个字符，其格式如下：

```
<$string>[<$index>]
```

或

```
<$string>{<$index>}
```

上面式子可以返回字符串$string 中指定索引位$index 上的字符，类型为字符串。如果指定的索引$index 无效，比如$index<0，或者$index>=strlen($string)，那么 PHP 系统将产生一条 Notice 错误信息，式子返回空串。

6.3.2 获取子串

substr 和 mb_substr 函数可以获取字符串中指定位置上的子串，语法格式如下：

```
string substr(string $str, int $start [, int $length])
string mb_substr(string $str, int $start [, int $length [, string $encoding]])
```

函数从参数字符串$str 中取出指定位置上的子串并返回，$start 指定子串的起始位置，$length 指定子串的长度。

说明：

（1）函数 substr 以字节为单位；函数 mb_substr 以字符为单位，$encoding 指定字符串采用的字符集的名称。下面描述统一以字符为单位，但实际上，就函数 substr 来说，应该是指字节。

（2）若$start 为非负整数，则子串的起始位置是字符串中下标为$start 的字符。比如$start 为 0，则起始位置就是字符串的首字符。若指定的子串起始位置超越了字符串的最后一个字符，则函数返回 false。

（3）若$start 为负整数，则子串的起始位置是字符串的倒数第-$start 个字符。比如$start 为-1，则起始位置就是字符串的最后一个字符。若指定的子串起始位置超越了字符串的首字符，那么子串的起始位置为首字符。

（4）若缺省$length，则子串从起始位置一直取到字符串的最后一个字符。

（5）若$length 为 0 或 false，则函数返回空串。

（6）若$length 为正整数，则其指定返回子串的字符个数；若该值太大，则取至字符串的最后一个字符。

（7）若$length 为负整数，则子串从起始位置取至倒数第-$length 个字符前的一个字符为止，即保留字符串末尾的-$length 个字符。

① 若 –$length 大于字符串的长度，则函数返回 false；

② 若要保留的字符中最左边第 1 个字符正好是起始位置字符，则函数返回空串；

③ 若要保留的字符越过了起始位置字符且$start 为负整数，则函数返回空串；

④ 若要保留的字符越过了起始位置字符且$start 为非负整数，则函数返回 false。

（8）在 substr 函数中，如果$length 为 NULL，那么函数返回空串。在 mb_substr 函数中，如果$length 为 NULL，那么子串从起始位置一直取至字符串的最后一个字符。

【例 6-3】 获取子串。代码如下：

```
<?php
echo substr("abcdef", -1)."<br/>";      // 输出：f
echo substr("abcdef", -2)."<br/>";      // 输出：ef
echo substr("abcdef", -3, 1)."<br/>";   // 输出：d
echo substr("abcdef", 0, -1)."<br/>";   // 输出：abcde
echo substr("abcdef", 2, -1)."<br/>";   // 输出：cde
echo substr("abcdef", -3, -1)."<br/>";  // 输出：de
var_dump(substr("abcdef", -3, -4));     // 输出：string(0) ""
var_dump(substr("abcdef", 4, -4));      // 输出：bool(false)

$str = "PHP 语言";
echo substr($str, 3, 3)."<br/>";        // 输出：语
echo mb_substr($str, 3, 2, "UTF-8");    // 输出：语言
?>
```

6.3.3 查找子串

查找子串是指在字符串中定位指定的子串，返回子串首字符在字符串中的位置。用于查找子串的函数包括 strpos、stripos、strrpos 和 strripos。

1. strpos 函数

该函数的语法格式如下：

```
mixed strpos(string $haystack, mixed $needle [, int $offset=0])
```

说明：

（1）函数返回子串$needle 在字符串$haystack 中第一次出现的位置。若在字符串中没有发现该子串，函数返回 false。

（2）$offset 指定搜索子串的起始位置，只能是非负整数，默认值为 0。

2. stripos 函数

该函数的语法格式如下：

```
mixed stripos(string $haystack, mixed $needle [, int $offset])
```

该函数在功能上与 strpos 函数相似，唯一不同的是，该函数在搜索时不区分字母的大

小写。

3. strrpos 函数

该函数的语法格式如下：

```
mixed strrpos(string $haystack, mixed $needle [, int $offset=0])
```

说明：

（1）函数返回子串$needle 在字符串$haystack 中最后一次出现的位置。若在字符串中没有发现该子串，函数返回 false。

（2）参数$offset 确定搜索子串的范围。该参数值可以是非负整数，也可以是负整数，默认值是 0。若是非负整数，则从下标为$offset 的字符开始查找，直至最后一个字符；若是负整数，从首字符（下标为 0）开始查找，直到倒数第-$offset 个字符为止（只要子串的首字符在此位置上即可）。

4. strripos 函数

该函数的语法格式如下：

```
mixed strripos(string $haystack, mixed $needle [,int $offset])
```

该函数在功能上与 strrpos 函数相似，唯一不同的是，该函数在搜索时不区分字母的大小写。

【例 6-4】 查找子串。假设 PHP 文件采用 UTF-8 字符集。代码如下：

```
<?php
$str = "MySQL 数据库 MySQL 数据库";
$str1 = "SQL";
var_dump(strpos($str, $str1));          // 输出：int(2)
var_dump(strpos($str, $str1, 3));       // 输出：int(16)

var_dump(strrpos($str, $str1));         // 输出：int(16)
var_dump(strrpos($str, $str1, 17));     // 输出：bool(false)
var_dump(strrpos($str, $str1, -13));    // 输出：int(2)
var_dump(strrpos($str, $str1, -12));    // 输出：int(16)
?>
```

6.3.4 替换子串

替换子串是指用特定内容（称为替换串）替换字符串中指定的子串。能实现子串替换功能的函数包括 str_replace 和 substr_replace。

1. str_replace 函数

实现子串替换，其中字符串、子串和替换串都由参数来指定。其语法格式如下：

```
mixed str_replace(mixed $search, mixed $replace, mixed $subject [, int
&$count])
```

函数用$replace（替换串）替换$subject（字符串）中出现的所有$search（子串），返回

替换后的结果字符串。

说明：

（1）如果参数$search 和$replace 都是数组，那么就依次用$search 中的各个元素（子串）和$replace 中的对应元素（替换串），对$subject（字符串）进行子串替换操作。如果$replace 的元素比$search 的元素少，那么用空串作为$search 中多余元素的替换串。

（2）如果参数$search 是数组，而参数$replace 是字符串，那么$replace 被用作$search 中的每个元素（子串）的替换串。

（3）如果参数$subject 是一个数组，则上述子串替换操作将依次执行于$subject 数组的每个元素，函数返回一个数组。

（4）如果指定$count 变量，那么在函数返回时，该变量将保存着执行替换操作的次数。

【例 6-5】 用 str_replace 函数实现子串替换。代码如下：

```
<?php
$str = "Line 1\nLine 2\rLine 3\r\nLine 4";
$order = array("\r\n", "\n", "\r");
$replace = '<br />';
echo str_replace($order, $replace, $str)."<br/>";
echo "------------------<br />";
$search = array('A', 'B', 'C', 'D', 'E');
$replace = array('B', 'C', 'D', 'E', 'F');
$subject = 'A';
echo str_replace($search, $replace, $subject)."<br/>";
echo "------------------<br />";
$letters = array('a', 'p');
$fruit = array('apple', 'pear');
$text = 'a p';
$output = str_replace($letters, $fruit, $text, $count);
echo $output."<br/>";
echo $count."<br/>";
?>
```

下面是代码运行的呈现结果：

```
Line 1
Line 2
Line 3
Line 4
------------------
F
------------------
apearpearle pear
4
```

2. substr_replace 函数

实现子串替换，其中要被替换的子串由其起始位置和长度确定。其语法格式如下：

```
mixed substr_replace(mixed $str, mixed $replacement, mixed $start[, mixed
$length])
```

函数用$replacement 替换字符串$str 中从$start 开始、长度为$length 的子串，返回替换后的结果字符串。

说明：

（1）参数$start 指定要被替换的子串的起始位置。若$start 为非负整数，子串的起始位置是下标为$start 的字符；若$start 为负整数，子串的起始位置是倒数第–$start 个字符。

（2）参数$length 指定要被替换的子串的长度。若$length 为正整数，则其指定子串的字符个数；若$length 为负整数，则子串从起始位置取至倒数第–$length 个字符前的一个字符为止；若$length 为 0，函数的功能是插入，即在起始位置之前插入替换串。若缺省$length，则子串从起始位置取至最后一个字符。

（3）如果参数$str 是数组，则$replacement、$start 和$length 会依次应用于$str 中的每个元素。如果$replacement、$start 和$length 也是数组，那么包括$str 在内的各数组对应元素分别进行替换操作。此时，函数返回一个数组。

【例 6-6】 用 substr_replace 实现子串替换。代码如下：

```
<?php
$var = 'ABCDEFGH:/MNRPQR/';
echo substr_replace($var, 'bob', 0)."<br/>";   //输出:bob
echo substr_replace($var, 'bob', 0, 0)."<br/>";//输出:bobABCDEFGH:/MNRPQR/
echo substr_replace($var, 'bob', 10, -1)."<br/>";//输出:ABCDEFGH:/bob/
echo substr_replace($var, 'bob', -7, -1)."<br/>";//输出:ABCDEFGH:/bob/
echo substr_replace($var, '', 10, -1)."<br/>"; //输出:ABCDEFGH://
?>
```

6.4 分割和连接字符串

分割字符串是指将一个字符串分割成若干个子串；连接字符串是指将若干字符串连接成一个新的字符串。

1. explode 函数

explode 函数实现字符串分割，其语法格式如下：

```
array explode(string $delimiter, string $string [, int $limit])
```

函数依据定界字符串$delimiter 将字符串$string 分割成若干子串，并将各子串保存在一个数组中返回。

说明：

（1）若$limit 指定为正整数，则返回的数组最多包含$limit 个元素，其中最后一个元素包含$string 的剩余部分；若$limit 指定为 0，被当作 1 处理，此时返回的数组仅包含 1 个元

素，即字符串$string 本身；若$limit 指定为负整数，则返回的数组包含除了最后–limit 个子串外的所有子串，可能会不包含任何元素。

（2）若$delimiter 是空串，会给出一个警告（Warning）信息，函数返回 false。

2. implode 函数

implode 函数实现字符串连接，其语法格式如下：

```
string implode([string $glue,] array $pieces)
```

函数将参数数组$pieces 中的各元素值连接成一个字符串返回，两个元素值之间用参数$glue 连接。$glue 的默认值是空串。

【例 6-7】 字符串的分割与连接。代码如下：

```
<?php
$str = 'one|two|three|four';

$arr = explode('|', $str);
print_r($arr);                    // 输出: Array ( [0] => one [1] => two [2] => three [3] => four )
echo "<br/>";
print_r(explode('|', $str, 2));// 输出: Array ( [0] => one [1] => two|three|four )
echo "<br/>";
print_r(explode('|', $str, 1));//输出:Array([0] => one|two|three| four )
echo "<br/>";
print_r(explode('|', $str, -1));// 输出: Array ( [0] => one [1] => two[2] => three )
echo "<br/>";

echo implode(":", $arr);          // 输出: one:two:three:four
?>
```

6.5 格式化输出

格式化输出是指按照指定的格式要求，将各种类型数据转换成字符串并输出。这里介绍相关的两个函数 printf 和 sprintf。

1. printf 函数

printf 函数可以格式化指定数据并输出，其语法格式如下：

```
int printf(string $format [, mixed $arg1 [, mixed $arg2]*])
```

函数根据格式化模板$format 对其他参数数据（$arg1、$arg2、…）进行格式化产生一个字符串并输出。函数返回格式化产生的字符串的字节长度。

说明：

（1）格式化模板由普通字符和格式符组成。普通字符原样出现在结果串中，格式符通过应用于对应的参数数据出现在结果串中。

（2）格式符的一般语法格式如下：

```
%[+]['<填充符>][-][<宽度>][.<精度>]<类型>
```

默认情况下，正数不输出正号，负数输出负号（–）。如果指定符号“+”（正号），正数也会输出正号（+）。

默认情况下，空格会填充多余的空间以满足指定的宽度。可以指定其他字符填充多余的空间。除了空格和 0，其他填充符必须以“'”（单引号）作为前缀。

默认情况下，数据在指定的宽度内右对齐。如果指定符号“–”（负号），则数据在指定宽度内左对齐。

精度以小数点开始。对于浮点数来说，精度指明了小数点后面要显示的位数。对于字符串来说，精度指明保留多少个字符（字节），后面的字符被截掉。

（3）格式符中的类型指明该格式符如何应用于参数数据，各种类型及其意义如表 6-1 所示。

表 6-1　格式符中类型及其意义

类型	意　　义
b	解释为整数，表示为相应的二进制形式
c	解释为整数，作为一个字符的 ASCII 码，表示为相应的字符（宽度无效）
d	解释为整数，表示为一个有符号十进制数
f	解释为浮点数
o	解释为整数，表示为相应的八进制形式
s	解释为字符串
u	解释为整数，表示为一个无符号十进制数。例如，–1 表示为 4294967295
x	解释为整数，表示为相应的十六进制形式（使用小写字母 a~f）
X	解释为整数，表示为相应的十六进制形式（使用大写字母 A~F）

2. sprintf 函数

sprintf 函数可以格式化指定数据并返回，其语法格式如下：

```
string sprintf(string $format [, mixed $arg1 [, mixed $arg2]*])
```

函数根据格式化模板$format 对其他参数数据（$arg1、$arg2、…）进行格式化产生一个字符串并返回。与 printf 函数不同，sprintf 函数不产生输出。

【例 6-8】 格式化输出。代码如下：

```
<?php
$num = 50;
$location = 'tree';
$format = "There are %d monkeys in the %s<br />";
printf($format, $num, $location);

$price = 12.567;
```

```
echo sprintf("The price is %5.2f", $price);
?>
```

下面是代码运行的输出结果：

```
There are 50 monkeys in the tree
The price is 12.57
```

6.6　字符串特殊处理

下面是一些常用的对字符串进行特殊处理的函数：

（1）nl2br（将新行符转换成 HTML 换行元素）；

（2）htmlspecialchars（将特殊字符转换成 HTML 实体）；

（3）urlencode（对 URL 进行编码）；

（4）addslashes（在特殊字符前加反斜杠）。

1. nl2br 函数

该函数的语法格式如下：

```
string nl2br(string $string)
```

函数将参数字符串$string 中每个新行符（\n、\r、\n\r 或\r\n）转换为 HTML 的换行元素（
），产生新的字符串返回。

说明：在浏览器中，普通意义上的新行符会被忽略（通常仅显示一个空格）。转换成
可以达到换行的效果。

2. htmlspecialchars 函数

该函数的语法格式如下：

```
string htmlspecialchars(string $string [, int $quotestyle [, string $encoding
[, bool $double_encode]]])
```

函数将参数字符串$string 中的一些特殊字符转换成 HTML 实体，以使这些字符不作为 HTML 的语法成分被浏览器处理，而是在浏览器上原样呈现。表 6-2 列出了特殊字符及其相应的 HTML 实体。

表 6-2　特殊字符及其相应的 HTML 实体

特殊字符	HTML 实体
&	&
"	"
'	'
<	<
>	>

说明：

（1）参数$quotestyle 指定是否对单引号和双引号进行转换，可取的值有以下 3 种。

① ENT_COMPAT：默认值。仅转换双引号。

② ENT_QUOTES：双引号和单引号都转换。

③ ENT_NOQUOTES：双引号和单引号都不转换。

（2）参数$encoding 指定字符串所用的字符集名称，比如：ISO-8859-1、UTF-8、GBK 等。

（3）参数$double_encode 指定是否对已有的 HTML 实体进行转换，默认值为 true，即要转换。

【例 6-9】 使用 nl2br 与 htmlspecialchars 函数。代码如下：

```
<?php
$str = nl2br("Welcome\r\nBEIJING");
echo $str;       // Welcome<br />BEIJING
echo "<br />";

$old = "<span>This's 语言</span>";
echo $old;       // <span>This's 语言</span>
echo "<br/>";
$new = htmlspecialchars($old);
echo $new;       // &lt;span&gt;This's 语言&lt;/span&gt;
echo "<br />";
$new1 = htmlspecialchars($new);
echo $new1;      // &amp;lt;span&amp;gt;This's 语言&amp;lt;/span&amp;gt;
echo "<br />";
$new2 = htmlspecialchars($new, ENT_QUOTES, "utf-8", false);
echo $new2;      // &lt;span&gt;This&#039;s 语言&lt;/span&gt;
?>
```

下面是代码运行的呈现结果：

```
Welcome
BEIJING
This's 语言
<span>This's 语言</span>
&lt;span&gt;This's 语言&lt;/span&gt;
<span>This's 语言</span>
```

3. addslashes 函数

该函数的语法格式如下：

```
string addslashes(string $str)
```

函数在参数字符串$str 中的一些特殊字符前加上反斜杠（\）产生一个新的字符串返回。这里，特殊字符包括：单引号（'）、双引号（"）、反斜线（\）与空字符（NUL）。

这一功能有时在构建数据库的 SQL 语句时是非常有用的。

4. urlencode 函数

该函数的语法格式如下：

```
string urlencode(string $str)
```

函数对参数字符串$str 进行 URL 编码并返回。经常用于对作为 URL 中查询参数值的字符串进行相应的编码，以便作为 URL 的一部分。

通常，URL 只能包含字母数字以及“-”和“_”字符，对其他字符都要进行相应的编码。空格被编码为加号（+），其他字符被编码为一组或多组%跟两个十六进制数字的字符序列。在用户提交表单的场景，浏览器会自动对表单数据进行编码。当要利用 PHP 代码构建一个包含请求参数的 URL 时，可以利用该函数对请求参数值进行编码。

【例 6-10】 addslashes 与 urlencode 函数的使用。代码如下：

```
<?php
$str = "This's a cat";
echo "\$str='$str'";
echo "<br />";
$str = addslashes("This's a cat");
echo "\$str='$str'";
echo "<br />";

$book = "PHP&MySql Web 应用";
$book = urlencode($book);   // PHP%26MySql+Web%E5%BA%94%E7%94%A8
echo "<a href='6-10-p.php?i1=$book'>YES</a>";
?>
```

下面是代码运行的呈现结果：

```
$str='This's a cat'
$str='This\'s a cat'
YES
```

当单击超链接文本 YES 时，将请求名为 6-10-p.php 的 PHP 文件并携带一个名为 i1 的请求参数。假设 6-10-p.php 文件的代码如下：

```
<?php
$book = $_GET['i1'];
echo $book;
?>
```

那么代码运行的输出结果如下：

```
PHP&MySql Web 应用
```

6.7 正则表达式

正则表达式（Regular Expression）是一个从左到右匹配目标字符串的模式，可以用来验证某个字符串或子串是否具有特定的格式，或者执行复杂的子串搜索和替换操作。目前，很多工具软件、应用软件、编程语言和脚本语言都支持正则表达式。

PHP 支持两种正则表达式：Perl 兼容正则表达式（Perl Compatible Regular Expressions, PCRE）和 POSIX（Portable Operating System Interface of UNIX）扩展正则表达式。这里介绍 Perl 兼容正则表达式及其使用。

一个 PCRE 正则表达式（模式）放置在定界符之间，前后定界符要一致，可以使用除字母、数字、反斜杠、空白字符之外的字符。 经常使用的定界符是斜杠（/）。例如：/food/是一个模式，可以匹配所有包含 food 的字符串。

6.7.1 字符类

一个字符类在目标字符串中匹配一个单独的字符。字符类包括一般字符类和特殊字符类。

1. 一般字符类

一般字符类由一对方括号表示，方括号内列出可以匹配的字符集合，如一般字符类[abc]可以匹配字母 a、b 或 c。

方括号内开始处也可以出现符号“^”，表示“相反”“排除”的意思，此时方括号内应该列出不能匹配的任何字符，如一般字符类[^abc]可以匹配除字母 a、b 和 c 之外的任何字符。

方括号内还可以出现符号“-”，表示范围，如一般字符类[A-Z]可以匹配任何一个大写字母。表 6-3 给出了一般字符类的语法及其含义。

表 6-3　一般字符类

一般字符类	含　　义
[...]	字符集合。匹配所包含的任意一个字符。例如，'[abc]'可以匹配"plain"中的'a'
[^...]	反向字符集合。匹配未包含的任意字符。例如，'[^abc]'可以匹配"plain"中的'p'
[<char1>-<char2>]	字符范围。匹配指定范围内的任意字符。例如，'[a-z]'可以匹配任意小写字母
[^<char1>-<char2>]	排除式字符范围。匹配任何不在指定范围内的任意字符。例如，'[^a-z]'可以匹配任何非小写字母字符

2. 特殊字符类

特殊字符类除符号“.”之外，其他都以反斜杠（\）跟一个特定字母组成，表示相应的某类字符，例如“\d”表示数字类，可以匹配任何一个数字。

除字符“.”之外，其他特殊字符类既可以出现在方括号内（一般字符类），也可以出现在方括号外。字符“.”作为特殊字符类只能出现在方括号外，匹配除"\n"外的任何单个字符，如果出现在方括号内（一般字符类），则表示字符“.”本身。

特殊字符类及其含义如表 6-4 所示。其中，单词字符指的是任意字母、数字、下画线，也就是任意可以组成 Perl 单词的字符。

表 6-4　特殊字符类

特殊字符类	含　　义
.	出现在方括号外，匹配除"\n"外的任何单个字符。例如，"a.c"可以匹配 abc、acc、a2c、a-c、a#c 等
\w	匹配包括下画线的任何单词字符，等价于[A-Za-z0-9_]
\W	匹配任何非单词字符，等价于[^A-Za-z0-9_]
\d	匹配一个数字字符，等价于[0-9]
\D	匹配一个非数字字符，等价于[^0-9]
\s	匹配任何空白字符，包括空格、制表符、换页符等，但不包括\v，等价于[\n\r\f\t]。说明：方括号内最后包含一个空格
\S	匹配任何非空白字符，等价于[^\n\r\f\t]

【例 6-11】 使用字符类。代码如下：

```
<?php
echo preg_match("/111[abc]999/", "other111h999other");     // 输出：0
echo preg_match("/111[^a-z]999/", "other111W999other");    // 输出：1
echo preg_match("/111[\s.]999/", "other111s999other");     // 输出：0
echo preg_match("/111\s.999/", "other111\r\t999other");    // 输出：1
echo preg_match("/\w\d.\s\d/", "othera12 3other");         // 输出：1
echo preg_match("/\w\d.\s\d/", "othera12\s3other");        // 输出：0
?>
```

这里，preg_match 是一个 PHP 的模式匹配函数，其中第 1 个参数指定一个正则表达式，第 2 个参数指定要验证的字符串。若要验证的字符串与指定的正则表达式相匹配，函数返回 1，否则函数返回 0。

6.7.2　元字符与转义字符

元字符是正则表达式语言保留的、具有特定语法作用的字符。转义字符是以反斜杠（\）开头的字符序列，用以匹配某个字符。

1. 元字符

在正则表达式中，每个元字符有其特殊含义。如果要匹配这些字符本身，一般可以在这些字符前面加一个反斜杠（\）。元字符包括以下几种：

-　$　(　)　*　+　?　.　[　{　|　\　^

（1）元字符“-”用于在方括号内，表示字符范围。如果出现在方括号外，该字符没有特殊的语法作用，既可以加反斜杠，也可以不加反斜杠而直接使用该字符本身。

（2）元字符“$”“(”“)”“*”“+”“?”“.”“[”“{”“|”用于方括号外，其中符号“.”和“[”在前面已经介绍，其他符号会在后面陆续介绍。如果出现在方括号内，这些字符没有特殊的语法作用，既可以加反斜杠，也可以不加反斜杠而直接使用这些字符本身。

（3）元字符“\”“^”既可用于方括号外，也可用于方括号内，其中“\”表示转义字符，“^”在方括号内表示“相反”和“排除”的意思，在方括号外表示一个断言。无论在方括号外，还是在方括号内，一般可分别用“\\”或“\^”匹配这两个字符本身。

2. 转义字符

在字符串中，可以用以反斜杠（\）开头的转义字符来表示某个字符，特别是一些非打印字符，如转义字符“\n”表示换行符。

正则表达式也有转义字符的概念，但有所区别：一是它的转义字符不是表示某个字符，而是用于匹配某个字符；二是它的应用范围更广。首先，对任何一个非数字字母的字符，在正则表达式中都可以用在该字符前加上反斜杠产生的转义字符来匹配。这种应用包括之前介绍的元字符。其次，与字符串的转义字符一样，它用一些特殊的转义字符匹配一些非打印字符，如转义字符“\n”匹配换行符。表 6-5 给出了正则表达式中的转义字符。

表 6-5　正则表达式中的转义字符

转义字符	含　　义
\非数字字母字符	匹配该非数字字母字符
\f	匹配一个换页符，即\x0c
\n	匹配一个换行符，即\x0a
\r	匹配一个回车符，即\x0d
\t	匹配一个制表符，即\x09
\v	匹配一个垂直制表符，即\x0b
\e	匹配一个 Esc 字符，即\x1b
\<octal>	由 3 位八进制码指定的字符，如\101 匹配一个字母 A
\x<hex>	由 1 或 2 位十六进制码指定的字符，如\x41 匹配一个字母 A
\c<char>	匹配由<char>指明的控制字符，如\cM 匹配一个 Control-M

由于正则表达式也被表示成字符串，所以要区分字符串中的转义字符和正则表达式中的转义字符。在具体处理时，PHP 解释器会先把字符串中的转义字符转换为相应的字符，然后正则表达式解释器再用正则表达式中的转义字符去匹配目标字符串中相应的字符。比如，在一个正则表达式字符串中有 4 个反斜杠“\\\\”，PHP 解释器首先会将其转换成 2 个反斜杠，然后正则表达式解释器会再用其去匹配单个反斜杠。

【例 6-12】 转义元字符和字符。代码如下：

```
<?php
echo preg_match('/a\n\x30\[/', 'a\n0[');        // 输出：0
echo preg_match('/a\n\x30\[/', "a\n0[");        // 输出：1
echo preg_match("/a\n\x30\[/", 'a\n0[');        // 输出：0
echo preg_match("/a\n\x30\[/", "a\n0[");        // 输出：1
echo preg_match('/a\\\n\x30\[/', 'a\n0[');      // 输出：1
echo preg_match("/a\\\n\x30\[/", 'a\n0[');      // 输出：0
?>
```

这里解释一下第 6 行代码。第 6 行的正则表达式字符串是一个单引号字符串，经处理字符串中的转义字符后得到的正则表达式应该是“/a\\n\x30\[/”。然后正则表达式解释器会用它去匹配目标字符串。这里，转义字符“\\”匹配一个反斜杠，转义字符“\x30”匹配数字 0，转义字符“\[”匹配字符“[”。

6.7.3 选项模式与子模式

字符类、转义字符等用于匹配特定字符，而选项模式与子模式则属于模式的结构成分。

1. 选项模式

选项模式是指提供多个可选模式，只要有一个模式能够匹配即告成功。

选项模式用竖杠（|）来表示，即在两个可选模式之间用竖杠分隔。匹配的处理从左到右尝试每一个可选模式，并且使用第一个成功匹配的。

例如，模式"/com|edu|net/"可以匹配"com"或"edu"或"net"。

2. 子模式

子模式是模式的一部分。子模式以括号为定界符，并且可以嵌套。将一个模式中的一部分标记为子模式主要有以下两个作用。

（1）将选项模式局部化。例如，模式"/cat(arcat|erpillar|)/"可以匹配"cat"或"cataract"或"caterpilla"。注意，其中的子模式中有 3 个可选项，最后一个可选项为空。

（2）除了可以保存和获取整个模式的匹配结果，还能够保存和获取各子模式的匹配结果。

有关子模式的语法格式及含义如表 6-6 所示。

表 6-6 子模式的语法及其含义

子模式	含　义
(<pattern>)	形成子模式，所获取的子模式匹配结果会保存在相应的数组中。要匹配括号字符，可以使用“\(”和“\)”
(?:<pattern>)	形成子模式，但所获取的子模式匹配结果不会进行存储。或者说，该子模式是一个非捕获匹配，相应的匹配结果不需要保存以供之后使用
\<num>	后向引用，匹配之前第 num 个已保存的子模式匹配结果

不仅整个模式的匹配结果能够保存，各子模式的匹配结果也可以保存。但在实际应用中，并不是所有的子模式匹配结果都需要保存。在子模式定义的左括号后面紧跟字符串"?:"，会使得该子模式的匹配结果不被保存，称为非捕获子模式。

在匹配过程中，先有子模式的匹配，后有整个模式的匹配。所谓后向引用是指引用之前保存下来的某个子模式匹配结果来匹配当前的目标内容。在后向引用中，num 应是一个大于 0 的整数，且之前至少有 num 个已保存的子模式匹配结果。

【例 6-13】 选项模式与子模式。代码如下：

```
<?php
echo preg_match('/industry|industries/', 'industries');    // 输出：1
echo "<br />";

```

```
5.  $ret = preg_match('/industr(y|ies)/', '123 industries 456', $result);
6.  print_r($ret);       // 输出: 1
7.  echo "<br />";
8.  print_r($result);    // 输出: Array ( [0] => industries [1] => ies )
9.  echo "<br />";
10.
11. $ret = preg_match('/(?:red|white) (king|queen)/', 'the white queen',
    $result);
12. print_r($ret);       // 输出: 1
13. echo "<br />";
14. print_r($result);    // 输出: Array ( [0] => white queen [1] => queen )
15. echo "<br />";
16.
17. $ret = preg_match('/(sens|respons)e and \1ibility/', 'sense and
    sensibility', $result);
18. print_r($ret);       // 输出: 1
19. echo "<br />";
20. print_r($result);    // 输出: Array ( [0] => sense and sensibility [1] =>
    sens )
21. ?>
```

说明：

（1）第 5 行代码中，preg_match 函数的参数不仅包含模式和目标字符串，还包括第 3 个参数$result。参数$result 用于存储模式和子模式的匹配结果，其类型是一个数组，其中第 1 个元素（索引为 0）保存着整个模式的匹配结果，后面各元素根据子模式出现的先后次序依次保存各子模式的匹配结果。若子模式出现嵌套，那么外部子模式的匹配结果保存在前面，内部子模式的匹配结果保存在后面。

（2）第 11 行代码中，模式"/(?:red|white) (king|queen)/"匹配目标字符串"the white queen"的结果将是如下数组：

array("white queen"、"queen")

其中，第 1 个元素是整个模式的匹配结果，第 2 个元素是第 2 个子模式的匹配结果。第 1 个子模式是一个非捕获匹配，其匹配结果没有被保存。

（3）第 17 行代码中，模式"/(sens|respons)e and \1ibility/"将会匹配"sense and sensibility"和"response and responsibility"，但不会匹配"sense and responsibility"。注意，模式中反斜杠后是数字 1，是一个后向引用。

6.7.4 量词

量词用于指定重复次数，出现在要重复的内容后面。可以重复的内容包括：

（1）单独字符；

（2）字符类；

（3）转义字符；

（4）后向引用；

（5）子模式。

表 6-7 列出了各种量词的语法及其含义。默认情况下，量词都是“贪婪”的，也就是说，它们会在不导致模式剩余部分匹配失败的前提下，尽可能多地匹配字符（直到最大允许的匹配次数）。当一个量词紧跟着问号（?）时，它就会成为“非贪婪”的， 它不再尽可能多地匹配，而是尽可能少地匹配。需要注意，问号本身也是量词，表示 0 次或 1 次。所以在这里，问号有两种含义及相应的用法。

表 6-7　量词

量词	含　义
*	匹配 0 次或更多次。例如，'xy*z'可以匹配"xz"、"xyz"、"xyyz"等。*等价于{0,}
+	匹配 1 次或更多次。例如，'xy+z'可以匹配"xyz"、"xyyz"等。+等价于{1,}
?	匹配 0 次或 1 次。例如，'do(es)?'可以匹配"do"和"does"。?等价于{0,1}
{<n>}	严格匹配 *n* 次。 例如，'xy{2}z'可以匹配"xyyz"
{<n>,}	至少匹配 *n* 次。 例如，'xy{2,}z'可以匹配"xyyz"、"xyyyz"、"xyyyyz"等
{<n>,<m>}	至少匹配 *n* 次、最多匹配 *m* 次。例如，'xy{2,3}z'可以匹配"xyyz"、"xyyyz"
?	该字符紧跟在其他量词（*、+、?、{*n*,}、{*n*,*m*}）后面时，形成“非贪婪”匹配

【例 6-14】量词的使用。代码如下：

```
<?php
$ret = preg_match('/o+/', "123 goooooogle 456", $result, PREG_OFFSET_CAPTURE);
print_r($ret);        // 输出: 1
echo "<br />";
print_r($result);   // 输出: Array ([0] => Array ( [0] => ooooo [1] => 5 ) )
echo "<br />";

$ret = preg_match('/o{2}/', "123 goooooogle 456", $result, PREG_OFFSET_CAPTURE);
print_r($ret);        // 输出: 1
echo "<br />";
print_r($result);   // 输出: Array ( [0] => Array ( [0] => oo [1] => 5 ) )
echo "<br />";

$ret = preg_match('/o{2,6}/', "123 goooooogle 456", $result, PREG_OFFSET_CAPTURE);
print_r($ret);        // 输出: 1
echo "<br />";
print_r($result);   // 输出: Array ([0] => Array ( [0] => ooooo [1] => 5 ) )
echo "<br />";

$ret = preg_match('/o+?/', "123 goooooogle 456", $result, PREG_OFFSET_CAPTURE);
```

```
21. print_r($ret);        // 输出：1
22. echo "<br />";
23. print_r($result);     // 输出：Array ( [0] => Array ( [0] => o [1] => 5 ) )
24. echo "<br />";
25.
26. $ret = preg_match('/.*foo/', "xfooxxxxxfoo", $result, PREG_OFFSET_CAPTURE);
27. print_r($ret);        // 输出：1
28. echo "<br />";
29. print_r($result);     // 输出：Array ([0] => Array ([0] => xfooxxxxxfoo [1] => 0))
30. echo "<br />";
31.
32. $ret = preg_match('/.*?foo/', "xfooxxxxxfoo", $result, PREG_OFFSET_CAPTURE);
33. print_r($ret);        // 输出：1
34. echo "<br />";
35. print_r($result);     // 输出：Array ( [0] => Array ( [0] => xfoo [1] => 0 ) )
36. ?>
```

说明：

（1）该例中的 preg_match 函数的参数不仅包含模式、目标字符串以及存放匹配结果的$result 数组，还指定了第 4 个参数。如果将该参数指定为 PREG_OFFSET_CAPTURE，则结果数组$result 中的每个元素都是一个数组，这些数组都包含两个元素，其中第 1 个元素存放匹配的子串，第 2 个元素存放该子串在目标字符串中的位置。

（2）这里对第 26 行代码执行时的模式匹配作一个解释。这里，模式是'/.*foo/'，其中的符号 “.” 可以匹配任何非换行符字符，量词*是 “贪婪” 的。目标字符串是"xfooxxxxxfoo"。首先目标字符串中首字母 x 会与模式中的符号.匹配。接下来，目标字符串中的 foo 是与模式中的符号“.”依次匹配，还是与模式中的 foo 匹配？如果符号.的量词是“非贪婪”的(即.*?)，或者目标字符串中后面不再有 foo，那么目标字符串中的该 foo 就与模式中的 foo 匹配，匹配结果就此确定。该行代码中，符号.的量词是 “贪婪” 的，且目标字符串中后面还有 foo，所以目标字符串中前面的 foo 以及其后的 5 个 x 就会依次与符号.匹配，目标字符串中后面的 foo 才与模式中的 foo 匹配。

6.7.5 断言

一个断言是对当前匹配位置之前或之后的字符的一个声明，它本身不会匹配或消耗目标字符串中的任何字符。

断言可分为简单断言和复杂断言，这里介绍常见的几种断言。简单的断言代码有^、$、\b 和\B 等。复杂的断言以子模式方式编码，包括肯定式预查断言(?=<pattern>)和否定式预查断言(?!<pattern>)，如表 6-8 所示。

表 6-8　断言

断言	含　　义
^	匹配目标字符串的开始位置，例如模式'/^The/'表示目标字符串应该以'The'开头
$	匹配目标字符串的结尾位置，例如模式'/no$/'表示目标字符串应该以'no'结尾
\b	匹配单词边界，例如模式'/ic\b/'可以匹配"economic"或"economic."中的'ic'
\B	匹配非单词边界，例如模式'/ic\B/'可以匹配"which"中的'ic'
(?=<pattern>)	肯定式预查，检测子模式是否匹配，但不改变当前匹配点，即预查不消耗字符。这里，子模式是一个非捕获匹配，也就是说，该匹配结果不需要保存
(?!<pattern>)	否定式预查，检测子模式是否不匹配，但不改变当前匹配点，即预查不消耗字符。这里，子模式是一个非捕获匹配，也就是说，该匹配结果不需要保存

【例 6-15】 断言的应用。

```
1.  <?php
2.  $ret = preg_match('/^[a-zA-Z_]\w*$/', "x123_");
3.  print_r($ret);      // 输出：1
4.  echo "<br />";
5.
6.  $subject = "The backslash character has several uses";
7.  $ret = preg_match('/\bha/',$subject, $result, PREG_OFFSET_CAPTURE);
8.  print_r($ret);      // 输出：1
9.  echo "<br />";
10. print_r($result);   // 输出：Array ( [0] => Array ( [0] => ha [1] => 24 ) )
11. echo "<br />";
12.
13. $ret = preg_match('/Windows(?=95|98|NT|2000)\d/', "Windows2000T",
    $result);
14. print_r($ret);      // 输出：1
15. echo "<br />";
16. print_r($result);   // 输出：Array ( [0] => Windows2 )
17. echo "<br />";
18.
19. $ret = preg_match('/Windows(?!95|98|NT|2000)\d/', "Windows8", $result);
20. print_r($ret);      // 输出：1
21. echo "<br />";
22. print_r($result);   // 输出：Array ( [0] => Windows8 )
23. echo "<br />";
24. ?>
```

下面对第 13 行代码执行时的模式匹配过程做一解释。

这里，模式'/Windows(?=95|98|NT|2000)\d/'包含一个肯定式预查断言，其作用是目标字符串中的 Windows 是否能匹配模式中的 Windows，要看其后的内容是否是 95、98、NT 或 2000。由于目标字符串中 Windows 后面的内容是 2000，所以上述匹配是成功的。因为预查

并不消耗字符，所以接下来要看目标字符串中的 2 是否与模式相匹配。由于模式后续是“\d”，即要求是数字，所以匹配成功。

6.8 PHP 模式匹配函数

这里介绍几个利用 PCRE 正则表达式（模式）进行字符串或子串格式验证以及子串搜索和替换操作的 PHP 函数。

1. preg_grep 函数

该函数实现批量匹配，即可以验证多个字符串是否匹配指定模式，其语法格式如下：

```
array preg_grep(string $pattern, array $input [, int $flags])
```

函数返回一个数组，数组包含输入数组$input 中与指定模式$pattern 相匹配的元素。返回数组各元素使用原来数组$input 中相应元素的键名进行索引。

$flag 是一个可选参数。若将其设置为 PREG_GREP_INVERT，函数返回输入数组$input 中与正则表达式$pattern 不匹配的元素。

【例 6-16】 preg_grep 函数的使用。代码如下：

```
<?php
$input = array("apple", "apply", "like", "ambition");
$result1 = preg_grep('/^a/', $input);
print_r($result1);  // 输出: Array ( [0] => apple [1] => apply [3] => ambition )
echo "<br />";
$result2 = preg_grep('/^a/', $input, PREG_GREP_INVERT);
print_r($result2);  // 输出: Array ( [2] => like )
?>
```

2. preg_match 函数

该函数在上一节已经频繁使用。它能验证某个字符串是否匹配指定模式，并能返回匹配的结果及其位置。函数的语法格式如下：

```
int preg_match(string $pattern, string $subject [, array &$matches [, int $flags]])
```

说明：

（1）若参数$subject（目标字符串）与参数$pattern（模式）相匹配，函数返回 1，否则返回 0。

（2）如果提供参数$matches，则该参数将被所匹配的结果所填充。$matches[0]保存与整个$pattern 相匹配的子串，$matches[1]保存与$pattern 中第 1 个需捕获的子模式相匹配的子串，依此类推。

（3）$flags 是一个可选参数。如果将该参数指定为 PREG_OFFSET_CAPTURE，则参数数组$matches 中的每个元素将是一个包含两个元素的数组，其中第 1 个元素存放匹配的子

串，第 2 个元素存放该子串在目标字符串$subject 中的位置。

3. preg_match_all 函数

该函数实现全范围匹配，即可以在目标字符串中找出能与指定模式匹配的所有子串。其语法格式如下：

```
int preg_match_all(string $pattern, string $subject [, array &$matches [,
int $flags]])
```

该函数的格式与函数 preg_match 相同，功能也类似。不同的是，preg_match 函数在发现第一个匹配结果时就停止搜索，而 preg_match_all 函数在找到第一个匹配结果后会继续搜索，直至搜索完整个目标字符串 subject。

该函数的参数$flags 可以取下面值。

（1）PREG_PATTERN_ORDER：该值是默认值。此时，$matches[0]是一个保存着所有与指定模式匹配的子串的数组，$matches[1]是一个保存着所有与第一个子模式匹配的子串的数组，依次类推。

（2）PREG_SET_ORDER：此时，$matches[0]是一个保存着第 1 次匹配时与指定模式及其各子模式相匹配的各子串的数组，$matches[1]是一个保存着第 2 次匹配时与指定模式及其各子模式相匹配的各子串的数组，依次类推。

（3）PREG_OFFSET_CAPTURE：此时每个匹配结果将保存在一个包含两个元素的数组中，其第 1 个元素保存着匹配串本身，第 2 个元素保存着该匹配串在目标字符串$subject 中的位置。该值可以和其他两个值组合使用，如：PREG_SET_ORDER|PREG_OFFSET_CAPTURE。

【例 6-17】 preg_match_all 函数的使用。代码如下：

```php
<?php
$pattern = '/ax.y-(13|24)b/';
$subject = "startax/y-24btttaxuy-13bend";
preg_match_all($pattern, $subject, $result, PREG_SET_ORDER);
print_r($result);
echo "<br />";
preg_match_all($pattern,$subject,$result,PREG_SET_ORDER|PREG_ OFFSET_
CAPTURE);
print_r($result);
?>
```

下面是代码运行的输出结果：

```
Array ( [0] => Array ( [0] => ax/y-24b [1] => 24 )
       [1] => Array ( [0] => axuy-13b [1] => 13))
Array ( [0] => Array ([0] => Array ([0] => ax/y-24b [1] => 5 )
       [1] => Array ([0] => 24 [1] => 10))
       [1] => Array ([0] => Array ([0] => axuy-13b [1] => 16 )
       [1] => Array ([0] => 13 [1] => 21)))
```

4. preg_replace 函数

该函数实现匹配并替换，其语法格式如下：

```
mixed preg_replace(mixed $pattern, mixed $replacement, mixed $subject [, int
$limit])
```

函数在目标字符串$subject 中搜索与模式$pattern 相匹配的所有子串，并用$replacement 替换，函数返回替换后新的字符串。

说明：

（1）如果指定参数$limit，则对每一对$subject 和$pattern 至多替换$limit 个匹配串，如果省略参数$limit 或将其设置为–1，则替换所有的匹配串。

（2）参数$pattern 可以是字符串也可以是字符串数组。如果是字符串数组，则目标字符串会针对其中的每个模式依次进行搜索、匹配和替换操作。

（3）参数$replacement 可以是字符串也可以是字符串数组。如果该参数是字符串，而参数$pattern 是一个模式数组，则针对每个模式的所有匹配项都由该字符串替换。如果该参数和参数$pattern 都是数组，则针对某个模式的所有匹配项都由该参数数组中对应的元素替换。如果该参数数组的元素个数小于参数$pattern 数组的元素个数，则针对额外模式的所有匹配项都由空串替换。

（4）参数$subject 可以是字符串也可以是字符串数组。如果该参数是字符串数组，则对每个字符串元素都进行相应的搜索、匹配和替换过程。函数返回替换后新的字符串数组。

（5）在参数$replacement 中还可以包含$n 或${n}，它表示与第 n 个子模式相匹配的内容。也就是说，可以用匹配的内容本身作为替换内容。这里 n 的取值范围是 0~99。若为 0，则表示与整个模式相匹配的内容。

【例 6-18】 preg_replace 函数的使用。代码如下：

```
1.  <?php
2.  $patterns = array('/a/', '/b/', '/c/');
3.  $replacements = array("1", "2", "3");
4.  $subject ="abcxaybzc";
5.  $result = preg_replace($patterns, $replacements, $subject);
6.  print_r($result);   // 输出：123x1y2z3
7.
8.  $result = preg_replace('/aa(x.y)ee/', '|${1}|', "aaax-yeee");
9.  print_r($result);   // 输出：a|x-y|e
10. ?>
```

注意：在第 8 行代码中，如果用双引号字符串表示参数$replacement，则其中的$前应加反斜杠，即如"|\${1}|"。

5. preg_split 函数

该函数实现匹配并分割，其语法格式如下：

```
array preg_split(string $pattern, string $subject [, int $limit [, int
$flags]])
```

参数$pattern 指定定界字符串的模式。函数依据定界字符串将目标字符串$subject 分割成若干子串，并保存在一个数组中返回。

说明：

（1）如果指定参数$limit，则至多返回$limit 个子串，其中最后一个子串包含目标字符串中后续所有内容。如果省略参数$limit 或将其设置为–1，则返回所有的子串。

（2）该函数的参数$flags 可以取以下值（这些值可通过位运算符|组合）：

① PREG_SPLIT_NO_EMPTY：仅返回非空子串。

② PREG_SPLIT_DELIM_CAPTURE：除了由定界字符串分割产生的子串，定界字符串中与捕获子模式匹配的子串也会被返回。

③ PREG_SPLIT_OFFSET_CAPTURE：不仅返回子串，也返回该子串的位置。所以返回的结果数组的每个元素都是一个数组，这些数组包含两个元素，其中第 1 个元素保存着子串，第 2 个元素保存着子串在目标字符串中的位置。

【例 6-19】 preg_split 函数的使用。代码如下：

```
<?php
$subject1 ="Hi,i am a student";
$result1 = preg_split('/[\s,]+/', $subject1);
print_r($result1);
echo "<br />";
$subject2 ="cat | dog : elephant";
$result2 = preg_split('/\s+([|:]+)\s+/', $subject2, -1, PREG_SPLIT_DELIM_CAPTURE);
print_r($result2);
?>
```

下面是代码运行的输出结果：

```
Array ( [0] => Hi [1] => i [2] => am [3] => a [4] => student )
Array ( [0] => cat [1] => | [2] => dog [3] => : [4] => elephant )
```

习　题　6

1. 写出下面 PHP 代码运行的输出结果

（1）

```
// 假设代码文件采用 UTF-8 字符集
$str = "您好 hello\t\n";
echo strlen($str), mb_strlen($str), mb_strlen(trim($str));
```

（2）

```
$arr1 = array("img12.png", "img10.png", "img2.png", "img1.png");
usort($arr1, "strnatcmp");
print_r($arr1);
```

（3）

```
// 假设代码文件采用 UTF-8 字符集
$s = '12345 信息 xyz';
echo $s[1], $s[11], "<br />";
$s[$s[1]] = '2';
echo $s;
```

（4）

```
// 假设代码文件采用 UTF-8 字符集
$str = "PHP 教程 PHP 教程";
var_dump(strpos($str, "hp", 1), stripos($str, "hp", 2));
echo "<br />";
var_dump(strrpos($str, "HP"), strrpos($str, "HP", 11));
echo "<br />";
echo strrpos($str, "HP", -8);
```

（5）

```
$email="user@example.com";
$len = strlen($email);
$p = strpos($email, "@");
echo substr($email, $p+1), "-", substr($email, 0, -($len-$p));
```

（6）

```
$x = 'apple';
echo substr_replace ($x, 'x', 1, 2);
```

（7）

```
$search = array('A', 'B', 'C');
$replace = array('AB', 'BC', 'CD');
$subject = 'AB';
echo str_replace($search, $replace, $subject);
```

（8）

```
$str = 'one,two,three,four';
$arr = explode(',', $str);
echo implode("-", $arr);
```

（9）

```
$price = 12.567;
echo sprintf("<p>The price is %5.2f</p>", $price);
```

（10）

```
$str = "Line 1\nLine 2\rLine 3\r\nLine 4\n";
$str1 = nl2br($str);
$str2 = htmlspecialchars($str1);
echo $str2;          // 写出在浏览器上的呈现结果
```

（11）

```
$a = preg_match("/111[\w\D]999/", "111w111,999");
$b = preg_match("/a\n\x30\[/", 'a\n0[');
echo $a.$b;
```

（12）

```
preg_match('/(blue|white) (?:sky|ocean)/', 'the blue sky', $result);
print_r($result);
```

（13）

```
$a = preg_match('/x\d*y/', '123x123y123');
$b = preg_match('/x\d+y/', '123xd+y123');
echo $a.$b;
```

(14)

```
$a = preg_match('/^[a-zA-Z_]\w*$/', "^awww");
$b = preg_match('/^[a-zA-Z_]\w*$/', "_123_");
echo $a.$b;
```

（15）

```
$pattern = '/expression(?=s).{3}/';
$subject = 'the expression expressions Language';
preg_match($pattern, $subject, $result, PREG_OFFSET_CAPTURE);
print_r($result);
```

2. 根据要求写 PHP 代码

（1）有一字符串$str，请写代码去除字符串尾部的所有数字字符。

（2）有一个字符串$str，其值是用逗号分隔的一组单词。请写代码把各单词取出放入一个新创建的数组$arr 中。

（3）有一个数组$arr，其中每个元素存放着一个单词。请写代码把各单词取出并用分号将其连接成一个字符串$str。

（4）编写一个自定义函数 reverse。函数接收一个字符串（可能包含汉字），返回参数字符串反转后的字符串。如参数字符串为“上海自来水”，返回字符串应该是“水来自海上”。

（5）写出一个能匹配身份证号码（18 位）的正则表达式。

（6）写出一个能匹配电子邮件地址的正则表达式。

（7）有一个数组$arr，其中每个元素存放着一个单词。请利用 preg_grep 函数把所有以字母 a 开头、以字母 x 结尾、长度不超过 7 个字母的单词从数组$arr 取出并保存到一个新的数组$ret 中。

（8）有一个字符串$string，请利用 preg_replace 函数把该字符串中所有的 HTML 标签去除。

（9）有一个字符串$str，其值为一段英文（各单词之间可能会用空白符号、逗号、句点分隔）。请利用 preg_split 函数把其中各单词取出放入一个新的数组$arr 中。

3. 简答题

（1）以下代码是判断字符串中是否存在#符号，请问是否正确？若错误，请修改之。

```
if(strpos($str, "#")) { … }
```

（2）假设$s1="123xyz"，$s2="xyz123"，请写出下面各表达式的结果。

A) $s1.$s2　　B) $s1 + $s2

C) "{$s1}{$s2}"　　D) implode("", array($s1,$s2))

（3）与正则表达式/.**123\d/相匹配的选项有哪些？

A) ******123　　B) *****_1234　　C) ******1234　　D) _*1234

第 7 章　MySQL 数据库基础

本章主题：

- 登录与账户管理；
- 权限管理；
- 数据库的创建与删除；
- MySQL 数据类型；
- 表的创建与删除；
- 数据的插入、更新和删除；
- 查询。

MySQL 是一种开源的关系型数据库管理系统。MySQL 数据库软件通常运行在客户/服务器（C/S）方式，包括一个多线程的 SQL 服务器，可以接受客户端程序、实用工具以及应用程序编程接口的访问。MySQL 具有快速、可靠、可伸缩和易用的特点，具有广泛的用户群，如维基百科就是它的一个著名的用户。

MySQL 为包括 C、C++、Eiffel、Java、Perl、PHP、Python、Ruby 和 Tcl 在内的编程语言和脚本语言提供了相应的应用程序编程接口（API），使得用这些语言编写的应用程序可以方便地访问 MySQL 数据库。目前，MySQL 是最常用的一种与 PHP 组合来开发网站和 Web 应用的数据库软件。

本章首先介绍使用 MySQL 监控程序访问 MySQL 服务器的方法，以及账户和权限管理的知识和相关语句，然后介绍 MySQL 数据库和表的创建、MySQL 数据类型等内容，最后介绍数据的插入、更新、删除和查询等语句及使用。

7.1　登录与账户管理

MySQL 服务器软件带有许多工具程序，用以实现数据库的管理功能。其中最常用的是命令行客户端程序 mysql，也称为 MySQL 监控程序。利用该程序，可以交互式或批处理方式执行 SQL 语句，完成数据库的创建、维护和查询等任务。

这里介绍利用监控程序登录 MySQL 服务器的方法以及管理用户账户的相关语句。任何用户要访问 MySQL 服务器，应该有自己的账户。

7.1.1　登录 MySQL 服务器

要使用 MySQL 监控程序，首先需要登录 MySQL 服务器，确认用户身份。要登录 MySQL 服务器，可以在操作系统命令提示符窗口中输入以下 mysql 命令：

```
mysql -h <host> -u <user> -p
```

其中，host 指定 MySQL 服务器所在的主机名，user 是登录用户的用户名。若从本地登录，可省略-h <host>项。如果用户账户没有密码，可以省略-p 项。

输入命令，按 Enter 键后会提示输入密码，输入密码后按 Enter 键即可登录。在用户账户没有密码的情况下，若省略-p，那么按 Enter 键后即可直接登录；否则可在系统给出输入密码提示信息后，直接按 Enter 键登录。对于刚安装好的 MySQL 服务器，超级用户 root 是没有密码的。

下面命令演示了以超级用户身份从本地登录 MySQL 服务器的场景。首先在操作系统命令提示符状态下输入 mysql 命令，指定用户名。按 Enter 键后，提示输入密码。由于超级用户没有密码，所以直接按 Enter 键。如果一切正常，MySQL 服务器会返回有关连接成功和帮助使用的提示信息，最终将出现提示符 mysql>。

```
C:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.24 MySQL Community Server (GPL)
…
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

mysql>是 MySQL 监控程序的命令提示符状态，表明已在 MySQL 监控程序控制之下。此时，用户可以输入并执行 SQL 语句了。

在 MySQL 监控程序命令提示符状态下，可以输入 SQL 的 quit 语句，断开与 MySQL 服务器的连接，返回到操作系统命令提示符状态。

```
mysql>quit;
Bye
C:\xampp\mysql\bin>
```

在 UNIX 平台下，也可以直接输入 control-D 断开与 MySQL 服务器的连接。

说明：打开操作系统命令提示符窗口的两种方法：

（1）在“开始”菜单中选择“附件”|“命令提示符”选项；

（2）在 XAMPP 控制面板窗口中单击右侧的 Shell 按钮。

7.1.2　用户账户管理

要登录 MySQL 服务器，首先需要有一个账户。这里介绍用户账户的创建、重命名、修改密码和删除等操作。

1. 创建账户

可以使用 CREATE USER 语句来创建账户，其语法格式如下：

```
CREATE USER '<user_name>'[@'<host_name>'] [IDENTIFIED BY '<password>']
```

语句创建一个新的 MySQL 账户。账户名由用户名和用户所用机器的主机名组成，因此

来自不同客户主机的用户可以有相同的用户名，属于不同账户。

主机名可以是主机的域名，也可以是主机的 IP 地址。主机名可以使用通配符_（下画线）和%（百分号）。其中，_表示任意单个字符，%表示任意多个字符，如'%.mysql.com'、'192.168.1.%'。如果只指定用户名，没有指定主机名，那么等价于'<user_name>'@'%'，意味着该用户可以从任何客户机登录和访问服务器。

如果缺省 IDENTIFIED BY 子句，新建账户将是无密码的。用户在登录连接时，也不应该提供密码。当然，无密码的账户是不安全的，不值得提倡。

【例 7-1】 创建了一个用户名为 zhang、密码为 123456 的账户，该账户只能从本地登录。

```
CREATE USER 'zhang'@'localhost' IDENTIFIED BY '123456';
```

【例 7-2】 创建一个用户名为 liming 的无密码账户，该账户只能从本地登录。

```
CREATE USER 'liming'@'localhost';
```

通常，要使用 CREATE USER 语句创建一个 MySQL 新账户，当前用户需要有全局级的 CREATE USER 权限。

2. 重命名账户

可以使用 RENAME USER 语句重命名账户，其语法格式如下：

```
RENAME USER '<user_name1>'[@'<host_name1>']
         TO '<user_name2>'[@'<host_name2>']
```

语句对指定的账户重新命名。指定账户名时，如果只指定用户名，没有指定主机名，那么'%'作为主机名。

【例 7-3】 将账户'liming'@'localhost'更名为'ming'@'127.0.0.1'。

```
RENAME USER 'liming'@'localhost' TO 'ming'@'127.0.0.1';
```

通常，要使用 RENAME USER 语句重命名一个 MySQL 账户，当前用户需要有全局级的 CREATE USER 权限。

3. 修改账户密码

可以使用 SET PASSWORD 语句修改密码，其语法格式如下：

```
SET PASSWORD FOR '<user_name>'[@'<host_name>']=PASSWORD('<password>')
```

语句对指定账户重新设置密码。指定账户名时，如果只指定用户名，没有指定主机名，那么'%'作为主机名。

【例 7-4】 将账户'zhang'@'localhost'的密码重新设置为'mypass'。

```
SET PASSWORD FOR 'zhang'@'localhost' = PASSWORD('mypass');
```

要使用 SET PASSWORD 语句重新设置账户密码，当前用户应该具有在 mysql 数据库上的 UPDATE 权限。

也可以使用 mysqladmin 工具程序修改账户密码。该工具程序用于执行一些管理员的操作，应该在操作系统命令提示符状态下调用。用 mysqladmin 工具程序修改账户密码的格式

如下：

```
mysqladmin -u <user_name> -p password
```

输入命令按 Enter 键后，系统首先会提示输入原密码并按 Enter 键（若原来没有密码，可直接按 Enter 键），然后在系统提示下输入新密码并进行确认即可。

4. 删除账户

可以使用 DROP USER 语句删除一个 MySQL 账户，并删除该账户所具有的所有权限。该语句的语法格式如下：

```
DROP USER '<user_name>'[@'<host_name>']
```

语句删除指定的账户。指定账户名时，如果只指定用户名，没有指定主机名，那么'%'作为主机名。

【例 7-5】 删除名为'zhang'@'localhost'的账户。

```
DROP USER 'zhang'@'localhost';
```

通常，要使用 DROP USER 语句删除账户，当前用户需要有全局级的 CREATE USER 权限。

7.2 权 限 管 理

本节首先简单介绍 MySQL 权限系统的概念和原理，然后再介绍 MySQL 权限管理的相关语句。

7.2.1 MySQL 权限系统简介

一个账户（如超级用户）可以向另外的账户在数据库系统的某些项目上授予一些操作权限。例如授予某个账户在系统内所有数据库上的 SELECT 权限、授予某个账户在 MYDB 数据库中 STUDENT 表上的 INSERT 权限等。这里，SELECT、INSERT 是权限名称，“所有数据库”和“MYDB 数据库中 STUDENT 表”指明权限应用的级别。下面是权限可应用的几个级别。

（1）全局级：涉及服务器系统本身，或系统内所有的数据库。

（2）数据库级：涉及特定数据库，及该数据库内的所有项目，例如表、存储过程等。

（3）表级：涉及特定表，及该表中的所有列。

（4）列级：涉及特定列。

这几个级别大致上是一种嵌套关系，即全局级包含数据库级，数据库级包含表级，表级包含列级。或者说，权限在级别上具有嵌套的属性，例如“授予某个账户在系统内所有数据库（全局级）上的 SELECT 权限”意味着该用户账户可以查询所有数据库内所有表中所有列的数据，即该账户在数据库级、表级和列级上都具有 SELECT 权限。一般仅为系统管理员授予全局级上的权限。

但一种权限并不能随意应用于某个级别。一种权限可应用的级别往往与该权限本身的

性质有关。例如权限 DELETE 可应用于全局级，意味着被授权账户可以删除系统内任何一个数据库内任何一个表中的记录。该权限也可应用数据库级和表级，但不能应用于列级。修改列中数据需要权限 UPDATE。表 7-1 列出了一些常用的权限及其可应用的级别。

表 7-1　常用权限及其可应用的级别

权限类型	含　义	可应用的级别
CREATE USER	允许使用 CREATE USER，DROP USER，RENAME USER 和 REVOKE ALL PRIVILEGES 语句	全局
FILE	允许从文件读入数据或把数据写到文件	全局
PROCESS	使用 SHOW PROCESSLIST 可以看到所有运行着的线程	全局
RELOAD	允许使用 FLUSH 操作	全局
SHOW DATABASES	使用 SHOW DATABASES 可以看到所有数据库	全局
SHUTDOWN	允许使用 mysqladmin shutdown 关闭 MySQL 服务器	全局
CREATE TEMPORARY TABLES	允许使用 CREATE TEMPORARY TABLE 语句	全局、数据库
EVENT	允许使用事件	全局、数据库
LOCK TABLES	允许使用 LOCK TABLES 语句	全局、数据库
ALTER	允许使用 ALTER TABLE 语句	全局、数据库、表
CREATE	允许创建数据库和表	全局、数据库、表
CREATE VIEW	允许创建更改视图	全局、数据库、表
DELETE	允许使用 DELETE 语句	全局、数据库、表
DROP	允许删除数据库、表和视图	全局、数据库、表
GRANT OPTION	允许授予或撤销权限	全局、数据库、表
INDEX	允许创建和删除索引	全局、数据库、表
SHOW VIEW	允许使用 SHOW CREATE VIEW 语句	全局、数据库、表
TRIGGER	允许触发器操作	全局、数据库、表
INSERT	允许使用 INSERT 语句	全局、数据库、表、列
REFERENCES	允许创建外键	全局、数据库、表、列
SELECT	允许使用 SELECT 语句	全局、数据库、表、列
UPDATE	允许使用 UPDATE 语句	全局、数据库、表、列
ALL [PRIVILEGES]	特定级别上的所有权限（不包括 GRANT OPTION）	全局、数据库、表

这里，有一类权限被称为管理员权限，它们不涉及特定数据库，通常只针对服务器系统本身，如 SHUTDOWN、RELOAD 等。这类权限只能应用于全局级。

所有账户的权限信息保存在相应的权限表中，所有的权限表保存在系统的 mysql 数据库中。其中，全局级权限信息保存在 user 表中，数据库级权限信息保存在 db 表中，表级权限信息保存在 tables_priv 表中，列级权限信息保存在 columns_priv 表中。

每一个权限表都包含范围列和权限列。范围列决定表中一行的权限信息作用的范围或

上下文。比如，mysql.user 表的范围列包括 Host、User 和 Password，即涵盖了一个用户账户的完整信息（包括密码）。mysql.user 表中的一行对应着一个用户账户的全局级权限信息。又比如，mysql.db 表的范围列包括 Host、Db 和 User，不仅可以确定某个用户账户，还能够确定某个数据库。mysql.db 表中的一行涉及特定用户在特定数据库上的数据库级权限信息。表 7-2 描述了这 4 个权限表的大致结构。

表 7-2　权限表及其结构

表名	use	db	tables_priv	columns_priv
范围列	Host User Password	Host Db User	Host Db User Table_name	Host Db User Table_name Column_name
权限列	Select_priv Insert_priv Shutdown_priv …	Select_priv Insert_priv …	Table_priv Column_priv	Column_priv
其他列	…	…	…	…

权限表的权限列包含特定范围的权限信息。mysql.user 表的权限列比较多，每种允许全局级的权限都对应着一列，例如 Select_priv、Shutdown_priv 等。这些权限列都是 enum 类型的，可以取'Y'或'N'值，分别表示特定范围是否被授予全局级上的该权限。mysql.db 表的情况与 mysql.user 表类似，每种允许数据库级的权限（不包括管理员权限）都对应着一列。这些权限列也是取'Y'或'N'值。mysql.tables_priv 表和 mysql.columns_priv 表的权限列都比较简单。mysql.tables_priv 表的权限列包括 Table_priv 和 Column_priv 两列，mysql.columns_priv 表的权限列仅有 Column_priv 一列。这些权限列都是 SET 类型的，可以包含特定范围所具有的全部权限名称。

MySQL 服务器利用权限表中的信息来实现对用户的访问控制。访问控制一般分两个阶段：连接验证和请求验证。首先，MySQL 服务器利用 mysql.user 表中范围列的信息进行连接验证，即根据用户的 Host、User 和 Password 值决定是否允许其连接。其次，对已经连接的用户的后续请求进行权限验证。MySQL 服务器会依次从 user、db、tables_priv 和 columns_priv 表中读取相关的权限信息以判断本次请求是否是授权的。在验证过程中，如果现有的权限信息已能确定请求是授权的，那么就不需要再考虑其他的权限信息。例如，如果当前请求是一个查询操作（SELECT），而 User 表中该用户账户对应行的 Select_priv 列的值为'Y '，那么就可以确定该请求是授权的。此时，MySQL 服务器就不需要再考虑其他权限表中的权限信息了。对有些请求，MySQL 服务器也可能需要综合多个权限表中的权限信息来判断请求是否是授权的。

7.2.2　权限管理语句

创建好账户后，接下来需要为其授予所需的操作权限。之后也可以撤销他的某些权限。

这里介绍有关权限管理的 GRANT、REVOKE 和 SHOW GRANTS 语句。

1. GRANT 语句

可以使用 GRANT 语句向账户授予一定的操作权限，这些权限通常被指定应用于某个级别上。语句的语法格式如下：

```
GRANT <priv_type> [(<column_list>)] [,<priv_type> [(<column_list>)]]*
   ON <priv_level>
   TO '<user_name>'[@'<host_name>'] [,'<user_name>'[@'<host_name>']]*
   [WITH GRANT OPTION]
```

其中，priv_type 指定要授予的权限，如 SELECT、UPDATE、DELETE 等，可以指定多个权限，各权限之间用逗号分隔。TO 子句指定所列权限要授予哪些账户。

ON 子句指定权限应用的级别，其中 priv_level 有以下几种形式。

① *.*：是指所有数据库的所有表，是全局级的。

② <db_name>.*：指定数据库中的所有表，是数据库级的。

③ *：是指当前数据库中的所有表，是数据库级的。如果不存在当前数据库，将出错。

④ <db_name>.<tbl_name>：指定数据库中的指定表，是表级的。

⑤ <tbl_name>：是指当前数据库的指定表，是表级的。如果不存在当前数据库，将出错。

如果要将权限应用于列级，可以在权限名后的括号内列出相关列，各列之间用逗号分隔。注意，即使只有一个列，也必须要用括号括起来。另外，从语法格式上可以看出，各权限可以应用于不同的列。

如果选择 WITH GRANT OPTION 选项，那么被授权的指定账户就具有了在指定级别上的 GRANT OPTION 权限，即该账户可以把他所获得的该级别（或低级别）上的权限再授予其他账户（或撤销已授予的权限）。

注意：GRANT OPTION 权限没有列级权限，MySQL 只根据 ON <priv_level>子句确定该权限的级别。但与其他权限一样，GRANT OPTION 权限也具有嵌套的属性，所以如果当前账户具有表级的 GRANT OPTION 权限，且具有列级的权限（如 SELECT），那么他也可以将他所具有的列级的权限授予其他账户。

要使用 GRANT 语句向其他账户授予特定级别上的一些权限，当前账户需要有相应级别的 GRANT OPTION 权限。

【例 7-6】 授予账户'zhang'@'localhost'可以在数据库 elective_manage 上进行全部操作的权限。代码如下：

```
GRANT ALL
   ON elective_manage.*
   TO 'zhang'@'localhost'
   WITH GRANT OPTION;
```

【例 7-7】 授予用户账户'liming'@'localhost'在数据库 elective_manage 上进行数据操纵（select、insert、update、delete）和数据定义（create、drop）的权限。代码如下：

```
GRANT select, insert, update, delete, create, drop
  ON elective_manage.*
  TO 'liming'@'localhost';
```

2. REVOKE 语句

可以使用 REVOKE 语句撤销账户的某些操作权限，其语法格式如下。

格式 1：

```
REVOKE <priv_type> [(<column_list>)] [,<priv_type> [(<column_list>)]]*
  ON <priv_level>
  FROM '<user_name>'[@'<host_name>'] [,'<user_name>'[@'<host_name>']]*
```

其中的 REVOKE 语句撤销指定账户在指定级别上的指定权限。要使用该格式，当前账户必须在指定级别上具有 GRANT OPTION 权限及要撤回的权限。

格式 2：

```
REVOKE ALL [PRIVILEGES], GRANT OPTION
  FROM '<user_name>'[@'<host_name>'] [,'<user_name>'[@'<host_name>']]*
```

其中的 REVOKE 语句撤销指定账户的所有权限，包括 GRANT OPTION 权限。要使用该格式，当前账户必须具有全局级的 CREATE USER 权限或者具有对 mysql 数据库的 UPDATE 权限。

【例 7-8】撤销账户'liming'@'localhost'在数据库 elective_manage 上进行数据定义（create、drop）的权限。代码如下：

```
REVOKE create, drop
  ON elective_manage.*
  FROM 'liming'@'localhost';
```

【例 7-9】 撤销账户'liming'@'localhost'在各级别上的所有权限，包括 GRANT OPTION 权限。代码如下：

```
REVOKE ALL, GRANT OPTION
  FROM 'liming'@'localhost';
```

3. SHOW GRANTS 语句

可以使用 SHOW GRANTS 语句查看账户所具有的权限，其语法格式如下：

```
SHOW GRANTS [FOR '<user_name>'[@'<host_name>']]
```

语句显示一条或多条 GRANT 语句，表示执行这些 GRANT 语句可以授予指定账户目前所具有的权限。如果缺省 FOR 子句，语句显示当前账户的权限信息。

除非显示当前账户的权限信息，否则当前账户必须具有对 mysql 数据库的 SELECT 权限。

【例 7-10】 显示超级用户账户'root'@'localhost'的权限信息。代码如下：

```
SHOW GRANTS FOR 'root'@'localhost';
```

7.3　数据库的创建与删除

这里介绍如何创建、选择、查看和删除数据库。

7.3.1　创建数据库

创建数据库可以使用 CREATE DATABASE 语句，该语句的语法格式如下：

```
CREATE DATABASE [IF NOT EXISTS] <db_name>
   [[DEFAULT] CHARACTER SET [=] <charset_name>]
   [[DEFAULT] COLLATE [=] <collation_name>]
```

该语句创建一个指定名称的数据库。要使用该语句，当前用户需要具有对该数据库的 CREATE 权限。指定 IF NOT EXISTS 选项可使语句避免因试图创建已存在的数据库而引发的错误。

CHARACTER SET 子句指定数据库的默认字符集。字符集是字符的集合，包括每个字符的编码。COLLATE 子句指定数据库的默认排序规则。排序规则规定字符集内字符之间的大小比较以及各字符的排列次序，一个字符集可以有多种排序规则。数据库的这些特性设置被保存在数据库目录的 db.opt 文件中。

在 MySQL 中，每个数据库对应 MySQL 数据目录（data）下的一个同名目录。当数据库刚创建时，数据库内还没有表等对应的文件，数据库目录下只有 db.opt 文件。

【例 7-11】 创建了一个名为 elective_manage 的选课管理数据库，该数据库的默认字符集为 utf8，默认排序规则为 utf8_bin。代码如下：

```
CREATE DATABASE elective_manage CHARACTER SET = utf8 COLLATE = utf8_bin;
```

7.3.2　选择当前数据库

默认情况下，很多 SQL 语句的操作对象都是当前数据库（也称为默认数据库）。刚创建的数据库并不会自动成为当前数据库。使用 USE 语句可以选择指定的数据库为当前数据库。该语句的格式如下：

```
USE <db_name>
```

【例 7-12】 选择 elective_manage 为当前数据库。代码如下：

```
use elective_manage;
```

指定了当前数据库并不妨碍访问其他数据库的内容。下面代码访问 db1 数据库的 author 表以及 db2 数据库的 editor 表：

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

7.3.3 显示数据库列表

可以使用 SHOW DATABASES 语句显示服务器上数据库名的列表，其基本格式如下：

```
SHOW DATABASES
```

语句能否列出服务器上所有的数据库名，与当前用户账户的权限有关。如果用户账户具有全局级的 SHOW DATABASES 权限，语句就会列出服务器内所有的数据库。对于一般用户，语句只列出那些该用户账户具有某些操作权限的数据库。

说明：该语句只是简单列出 MySQL 数据目录（data）下的子目录名，所以列出的未必一定就是一个数据库名。

7.3.4 删除数据库

可以使用 DROP DATABASE 语句删除一个数据库，其语法格式如下：

```
DROP DATABASE [IF EXISTS] <db_name>
```

语句删除指定的数据库。指定 IF EXISTS 选项可使语句避免因试图删除不存在的数据库而引发的错误。

该语句可以删除当前数据库。删除当前数据库后，可以使用 USE 语句指定新的当前数据库。

使用这个语句必须谨慎，因为它将删除指定的整个数据库，包括该数据库的所有表及其数据。

7.4 MySQL 数据类型

在创建表时，需要为每列指定所需的数据类型。数据类型决定了数据的性质、取值范围和存储格式等。MySQL 提供了丰富的数据类型，大致包括数值型、日期和时间型以及字符串型三大类。

7.4.1 数值型

数值型数据分整数和实数。表达整数的类型包括 TINYINT（微整形）、SMALLINT（小整型）、MEDIUMINT（中整型）、INT（整型）和 BIGINT（大整型）。

表达实数的类型包括 DECIMAL（定点数）、FLOAT（单精度浮点数）和 DOUBLE（双精度浮点数）。表 7-3 列出了这些数值型数据的取值范围以及所需的存储字节。

表 7-3 MySQL 数值型

类　型	字节	取值范围
TINYINT	1	−128 ~ 127
TINYINT UNSIGNED	1	0 ~ 255
SMALLINT	2	−32768 ~ 32767

续表

类　　型	字节	取值范围
SMALLINT UNSIGNED	2	0～65535
MEDIUMINT	3	－8388608～8388607
MEDIUMINT UNSIGNED	3	0～16777215
INT	4	－2147483648～2147483647
INT UNSIGNED	4	0～4294967295
BIGINT	8	－9223372036854775808～9223372036854775807
BIGINT UNSIGNED	8	0～18446744073709551615
DECIMAL[(<M>[, <D>])] [UNSIGNED]	变长	M 是允许的总的数字位数（精度），最大值为 65。D 是允许的小数位数，最大值为 30。其中 M 指定的位数不包括小数点和符号 若缺省 D，默认值为 0。若缺省 M，默认值为 10 若指定 UNSIGNED，则不允许取负值
FLOAT [UNSIGNED]	4	$-3.402823466\times10^{38}$～$-1.175494351\times10^{-38}$ 0 $1.175494351\times10^{-38}$～$3.402823466\times10^{38}$ 若指定 UNSIGNED，则不允许取负值
DOUBLE [UNSIGNED]	8	$-1.7976931348623157\times10^{308}$～$-2.2250738585072014\times10^{-308}$ 0 $2.2250738585072014\times10^{-308}$～$1.7976931348623157\times10^{308}$ 若指定 UNSIGNED，则不允许取负值
BOOLEAN 或 BOOL	是 TINYINT 的同义词，false 被存储为 0，true 被存储为非 0 值（1）	

这里，各种整型和 DECIMAL 型是精确数值型，DOUBLE 型和 FLOAT 型是近似数值型。

对于每一种数值型，默认情况下都是有符号的，即其取值既可以是正值也可以是负值。如果指定 UNSIGNED，则成为无符号的，此时其取值只能是正值。

7.4.2　日期和时间型

涉及日期和时间的类型包括表示日期的 DATE 型，表示时间的 TIME 型，表示日期时间的 DATETIME 型和 TIMESTAMP 型，表示年份的 YEAR 型，如表 7-4 所示。

表 7-4　MySQL 日期和时间型

类　　型	字节数	取值范围
DATE	3	'1000-01-01'～'9999-12-31'
DATETIME	8 或 5	'1000-01-01 00:00:00'～'9999-12-31 23:59:59'
TIMESTAMP	4	'1970-01-01 00:00:01' UTC～'2038-01-19 03:14:07' UTC
TIME	3	'－838:59:59'～'838:59:59'
YEAR	1	'1901'～'2155'

1. 关于日期和时间型的一般说明

（1）与 DATETIME 型不同，TIMESTAMP 型保存的是 UTC 时间值（秒数）。当用户为 TIMESTAMP 型的列指定一个值时，MySQL 会先将其从当前时区转换为 UTC 然后再存储。当用户访问一个 TIMESTAMP 型列的值时，MySQL 会先将值由 UTC 转换为当前时区然后再返回。

（2）任何一种日期和时间型的列，都允许接收字符串或数值为其赋值。

- DATE：'2010-10-20'，20101020。
- DATETIME、TIMESTAMP：'2010-10-20 10:20:30'，20101020102030。
- TIME：'10:20:30'，102030。
- YEAR：'2010'，2010。

（3）TIME 型用于保存时间。既可以用来表示一天中的某个时间，也可以用来表示两个事件的间隔时间。可以用下面格式的文字表示一个 TIME 值：

```
'12:30:50'
'100:10:10'
'-100:10:10'
'4 4:10:10'     // 4 天又 4 小时 10 分 10 秒，相当于'100:10:10'
'-4 4:10:10'    // 相当于'-100:10:10'
```

（4）对于 DATE 和 DATETIME 型的数据，月份和日数为零是允许的，例如'1990-00-00'。这样的日期值在有些应用场景是有意义的。比如，知道一个人的生日年份，但不能确定哪月哪日。

（5）对于任何日期和时间型，都存在一个“零”值。对于 TIME 型，“零”值是它的值域范围内的一个值。对于其他日期和时间型，“零”值并不是它们值域范围内的一个值。

- DATE：'0000-00-00'。
- DATETIME、TIMESTAMP：'0000-00-00 00:00:00'。
- TIME：'00:00:00'。
- YEAR：0000。

当一个日期和时间型的值超出了它的值域范围或不合法时，如月份大于 12，分或秒大于 59，那么 MySQL 会将其保存为“零”值。

这里有一个例外，对于 TIME 型，如果一个值超出了它的值域范围，MySQL 会将其保存为其值域的最大值（'838:59:59'）或最小值（'－838:59:59'）。

2. TIMESTAMP 和 DATETIME 型的默认值

与其他数据类型不同，在定义 TIMESTAMP 型列时，如果缺省[NULL|NOT NULL]选项，则其默认设置是 NOT NULL，而非 NULL。

首先考虑 NULL 值的情况。当插入一行时，如果给某个 TIMESTAMP 或 DATETIME 型列指定为 NULL 值，则 MySQL 的处理规则如下：

（1）对于 TIMESTAMP 列，如果在定义列时没有指定 NULL，则自动取当前时间戳值；如果指定了 NULL，则取 NULL 值。

（2）对于 DATETIME 列，如果在定义列时没有指定 NOT NULL，则取 NULL 值；如果

指定了 NOT NULL，则出错。

然后再考虑默认值的情况。当插入一行时，如果没有给某个 TIMESTAMP 或 DATETIME 型列指定值，而该列在定义时又没有指定 DEFAULT 子句和 ON UPDATE 子句，则 MySQL 的处理规则如下：

（1）对于表中的第 1 个 TIMESTAMP 型列，如果在定义列时没有指定 NULL，则自动取当前时间戳值；如果指定了 NULL，则取 NULL 值。

（2）对于表中非第 1 个 TIMESTAMP 型列，如果在定义列时没有指定 NULL，则自动取"零"值；如果指定了 NULL，则取 NULL 值。

（3）对于 DATETIME 型列，如果在定义列时没有指定 NOT NULL，则取 NULL 值；如果指定了 NOT NULL，则自动取"零"值。

最后考虑自动初始化列与自动更新列的情况。所谓自动初始化列是指当插入一行而没有为该列指定值时，该列将自动设置为当前时间。默认情况下，表中的第 1 个 TIMESTAMP 型列是一个自动初始化列。这里的默认情况是指在定义列时没有指定 NULL，也没有指定 DEFAULT 子句和 ON UPDATE 子句。

在定义列时，通过指定 DEFAULT CURRENT_TIMESTAMP，可以将任何 TIMESTAMP 或 DATETIME 型列设置为自动初始化列。例如：

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

所谓自动更新列是指当更新某行其他列的值（确实有值发生改变）时，该列将自动更新为当前时间。默认情况下，表中的第 1 个 TIMESTAMP 型列也是一个自动更新列。

在定义列时，通过指定 ON UPDATE CURRENT_TIMESTAMP，可将任何 TIMESTAMP 或 DATETIME 型列设置为自动更新列。例如：

```
CREATE TABLE t2 (
  ts1 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,          # default 0
  ts2 TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP,     # default NULL
  dt1 DATETIME ON UPDATE CURRENT_TIMESTAMP,           # default NULL
  dt2 DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP   # default 0
);
```

这里，ts1 虽然是表中的第 1 个 TIMESTAMP 型列，但由于设置了 ON UPDATE 子句，其默认值不再是当前时间戳值，而是"零"值，所以它不是一个自动初始化列了。

当然，也可以将任何 TIMESTAMP 或 DATETIME 型列设置为既是自动初始化列也是自动更新列。例如：

```
CREATE TABLE t3 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

7.4.3 字符串型

字符串型包括 CHAR、BINARY、VARCHAR、VARBINARY、BLOB、TEXT、ENUM 和 SET，如表 7-5 所示。

表 7-5 MySQL 字符串型

类　型	描　述
CHAR[(<M>)]	固定长度为 M 个字符的字符串，其中 M 的取值范围为 0~255。若缺省 M，长度取 1，即 CHAR 相当于 CHAR(1)
BINARY[(<M>)]	固定长度为 M 个字节的二进制字符串，其中 M 的取值范围为 0~255。若缺省 M，长度取 1
VARCHAR(<M>)、VARBINARY(<M>)	最大长度为 M 个字符（字节）的可变长字符串或字节串。理论上，M 的取值范围为 0~65535，但受列数和行的总字节数限制
TINYBLOB、TINYTEXT	微 BLOB（字节串）和微 TEXT（字符串），最大支持 255 B 长度
BLOB、TEXT	BLOB（字节串）和 TEXT（字符串），最大支持 65 535B 长度
MEDIUMBLOB、MEDIUMTEXT	中 BLOB（字节串）和中 TEXT（字符串），最大支持 16 777 215B 长度
LONGBLOB、LONGTEXT	长 BLOB（字节串）和长 TEXT（字符串），最大支持 4 294 967 295B 长度
ENUM('member1', 'member2'…)	每个 ENUM 型预定义一个值列表，最多可包括 65 535 个不同值。ENUM 型列的值只能是值列表中的某个值。如果列声明允许 NULL，则 NULL 是默认值。如果列声明为 NOT NULL，则值列表的第 1 个值为默认值
SET('member1', 'member2'…)	每个 SET 型预定义一个值列表，多可包括 64 个不同值，每个值（字符串）本身不应该包含逗号（,）。SET 型列的值只能由值列表中的零个或多个值组成，两个值之间用逗号（,）分隔

除 ENUM 和 SET 外，其他字符串类型大致可分为非二进制字符串和二进制字符串两大类。非二进制字符串简称字符串，包括 CHAR、VARCHAR、TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。二进制字符串简称字节串，包括 BINARY、VARBINARY、TINTBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。

非二进制字符串类型的列具有字符集和排序规则的属性，即它们的字符取自特定或默认的字符集，它们的值能按特定或默认的排序规则排序。二进制字符串类型的列不具有字符集和排序规则的属性，其列值只被看作是字节串，对列值的比较和排序也只是根据其二进制码进行。

非二进制 CHAR 型和 VARCHAR 型的长度是以字符为单位的。例如 CHAR(5)表示长度是 5 个字符，VARCHAR(5)表示最大长度是 5 个字符。由于在不同的字符集中，字符编码所需的字节数是不一样的，比如一个汉字在有些字符集中用 2 字节编码，在另一些字符集中采用 3 字节编码，所以这两种类型列的实际列宽（字节数）往往大于指定的长度，且跟列所采用的字符集有关。

二进制 BINARY 型和 VARBINARY 型的长度是以字节为单位的。例如 BINARY(5)表示

长度是 5 个字节，VARBINARY(5)表示最大长度是 5 个字节。由于一个字符可能占用 1 B 长度，也可能占用多个字节，所以这两种类型列能够存放的字符数往往要少于指定的长度。

TEXT 可以被看作是一种 VARCHAR，BLOB 可以被看作是一种 VARBINARY。这里，TEXT、BLOB 与 VARCHAR、VARBINARY 之间的区别主要是实现技术和使用约束方面的不同。比如，TEXT 列和 BLOB 列的长度不受行的总宽度的限制，TEXT 列和 BLOB 列不能指定默认值等。各种 BLOB 之间和各种 TEXT 之间只是最大长度的不同，没有本质上的区别。

ENUM 型的值是一个字符串，这个字符串应该从一组允许的值中选择。一个 ENUM 型列可取的所有值必须在创建表时在列定义中枚举出来。例如：

```
CREATE TABLE shirts (
   name VARCHAR(20),
   size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);
```

实际存储时，ENUM 型列的每个值被自动存储为该值在这组枚举值中对应的索引号，如'x-small'被存储为 1。当读取时，这些索引号又会被自动转换回相应的字符串。下面演示了对 shirts 表的查询及显示的结果：

```
select * from shirts;
---------------------------------
 name   size
---------------------------------
 dress shirt    large
 t-shirt    medium
 polo shirt small
---------------------------------
```

如果查询时 ENUM 列的值出现在数值型上下文中，那么列值的索引号将被返回。下面代码演示了这种情况：

```
select name,size+0 from shirts;
---------------------------------
 name   size+0
---------------------------------
 dress shirt    4
 t-shirt    3
 polo shirt 2
---------------------------------
```

SET 型的值是一个字符串，这个字符串由预定义的一组值中的零个或多个值组成。各值之间用逗号分隔。这组值在创建表时在如下的列定义子句中列出来：

```
CREATE TABLE table1 (column1 SET('one', 'two', 'three', 'four'));
```

实际存储时，每个 SET 列值都会转换成一个数值存放，这里采用二进制编码的方式。

比如上面这个例子，一共有 4 个成员，那么可以用 4 个二进制位来编码，其中最低位对应第 1 个成员，依此类推。如果一个列值中包含第 1 个成员，则最低位为 1，否则为 0。所以如果一个列值为'one,three'，那么实际存储为数值 5，用二进制表示即为 0101。当读取时，这些数值又会被自动转换回相应的字符串。

7.5　表的创建与删除

一个数据库通常包含若干表，每个表可以保存有关的数据。创建表就是要定义表的结构，即规定该表每一列的列名、可以存放的数据的类型以及相关的属性和约束等。本节介绍表的创建、删除等操作。

7.5.1　创建表

下面从基本语法格式、定义列和定义表级约束三个方面介绍创建表的 SQL 语句的主要语法成分和功能。

1. 基本语法格式

创建表的 SQL 语句是 CREATE TABLE，其基本语法格式如下：

```
CREATE TABLE [<db-name>.]<table-name> (
   {<col_name> <column-definition>|<table-level-constraint>}
   [, {<col_name> <column-definition>|<table-level-constraint>}]*
)
```

语句在指定的数据库（db-name）中创建一个指定名称（table-name）的表。数据库名是可选项，若没有指定，将在当前数据库中创建表。如果指定的数据库不存在，或者没有指定数据库而当前数据库不存在，或者已经存在指定名称的表，那么语句出错。

2. 定义列

定义一个表的主要工作是定义表中各列，列定义（column-definition）的常用格式如下：

```
<data_type>
   [NULL | NOT NULL]
   [DEFAULT <default_value>]
   [AUTO_INCREMENT]
   [UNIQUE [KEY] | [PRIMARY] KEY]
```

其中，data-type 指定列的数据类型，是必选项。其他都是可选项，用于指定列的属性或约束。

（1）NULL 和 NOT NULL 用于指定列是否可以取 NULL 值，NULL 表示可以取 NULL 值，NOT NULL 表示不可以取 NULL 值。如果忽略该可选项，默认设置为 NULL。

（2）可以用 DEFAULT 子句为列指定默认值。这里，各种 BLOB 型和各种 TEXT 型的列不能指定默认值。

默认值 default_value 只能指定为常量，不能是函数或表达式。一个例外是，可以为 TIMESTAMP 型列指定默认值 CURRENT_TIMESTAMP。这里 CURRENT_TIMESTAMP 是一个 SQL 函数，它与 CURRENT_TIMESTAMP()和 NOW()函数具有相同的功能。

通常，当向一个表插入一行时，应该指定该行在每一列上的相应值。如果某列具有默认值或可以取 NULL 值，那么也可以不为该列指定值。此时，该列将取默认值；如果没有默认值，则取 NULL 值。

（3）整型和浮点型列可以指定 AUTO_INCREMENT，称为自增列。自增列必须是索引的，且每个表一般只能有一个自增列。当为自增列指定 NULL 或 0 值时，该列将设置为下一个序列值（通常为该列当前已有的最大值加 1）。自增列的序列值的初值为 1。

可以使用 SQL 函数 LAST_INSERT_ID()获取最后插入的行在自增列上的取值。

（4）如果一列指定为 UNIQUE KEY 或 UNIQUE，则将为该列建立一个唯一索引。对于唯一索引的列，除了 NULL 值，不允许存在其他相同的值。对于唯一索引的列，如果允许取 NULL 值，可以有多个 NULL 值。一个表可以包含多个唯一索引的列。

（5）如果一列指定为 PRIMARY KEY 或 KEY，则将为该列建立一个主索引。对于主索引的列（称为主键），不允许存在相同的值，且不允许取 NULL 值。如果主索引列没有指定 NOT NULL，则自动设置为 NOT NULL。一个表至多只能包含一个主索引。

如果主键由多列组成，则可以使用表级的 PRIMARY KEY (<列名表>)子句来进行定义。另外，可以使用 FOREIGN KEY…REFERENCES…子句为表定义外键。

3. 定义表级约束

有些数据完整性约束可以定义在列级，如单列主键。有些数据完整性约束则需要定义在表级，如多列主键。下面是定义表级约束的常用格式：

```
{
  PRIMARY KEY (<col_name> [,<col_name>]*)
  | UNIQUE [INDEX | KEY] [<index_name>] (<col_name> [,<col_name>]*)
  | {INDEX | KEY} [<index_name>] (<col_name> [,<col_name>]*)
  | CONSTRAINT <constraint_name> FOREIGN KEY (<col_name> [,<col_name>]*)
    REFERENCES [<db.name>.]<tbl_name> (<r_col_name> [,<r_col_name>]*)
       [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
       [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
}
```

说明：

（1）这里共有 4 个子句，每个子句都会创建一个索引。其中，PRIMARY KEY 子句创建一个主索引；UNIQUE INDEX 或 UNIQUE KEY 子句创建一个唯一索引；INDEX 或 KEY 子句创建一个普通索引，即非唯一索引；FOREIGN KEY 子句也会创建一个非唯一索引。

（2）每个索引可以指定一个名称。index_name 指定被创建索引的索引名。主索引的索引名总是 PRIMARY。

（3）每个索引可以针对一个列，也可以针对多个列。col_name[,col_name]*指定要创建索引的列。组成主键的所有列都必须是 NOT NULL。如果主键列没有指定 NOT NULL，则自动设置为 NOT NULL。

（4）FOREIGN KEY 子句用于定义数据的参照完整性约束。这里，constraint_name 指定约束名，col_name[,col_name]*指定参照列，r_col_name[,r_col_name]*指定被参照列。

在参照完整性约束中，参照列所在的表称为子表，在这里也就是当前正在创建的表。

被参照列所在的表称为父表，在这里由 tbl_name 指定。参照列称为外键，被参照列通常是父表的主键。

所谓数据的参照完整性约束是指子表中的一行在父表中应该有对应的行：子表中某行在参照列上的值与主表对应行在被参照列上的值应该相同。通常，子表中的一行对应主表中的一行，主表中的一行可以对应子表中的多行（或没有）。

为保证数据的参照完整性，服务器不允许在子表中插入（INSERT）或更新（UPDATE）行，使其在主表中没有对应的行。

子句 ON DELETE 和 ON UPDATE 指定了当删除主表中的行和更新主表中被参照列的值时，服务器应采取的动作。

① CASCADE：当删除主表中的行或更新主表中被参照列的值时，删除子表中所有对应行或相应更新子表中所有对应行在参照列上的值。

② SET NULL：当删除主表中的行或更新主表中被参照列的值时，将子表中所有对应行在参照列上的值设置为 NULL。

③ RESTRICT：对在子表中有对应行的主表行，不允许删除或更新其被参照列上的值。

④ NO ACTION：在 MySQL 中，NO ACTION 和 RESTRICT 有相同的含义。

如果不选 ON DELETE 子句和 ON UPDATE 子句，默认的动作是 RESTRICT。

在定义数据的参照完整性约束的同时，FOREIGN KEY 子句也会自动在参照列上创建一个与约束名（constraint_name）同名的非唯一索引。

7.5.2 创建表举例

这里以教务选课系统选课管理数据库 elective_manage 为例，说明表的创建和定义。该数据库包含学生表、教师表、课程表、开课表、学生选课表等 5 个表。

【例 7-13】 创建学生（student）表，其具体要求如表 7-6 所示。代码如下：

表 7-6 学生（student）表

列　　名	类　　型	说　　明
Sn	CHAR(12)	学号（用户名），主键
spassword	VARCHAR(12)	口令，不能取 NULL 值
sname	CHAR(4)	姓名，不能取 NULL 值
gender	CHAR	性别（男或女）
birthday	DATE	出生日期
email	VARCHAR(28)	邮箱地址，不能取 NULL 值

```
CREATE TABLE student (
  sn CHAR (12) PRIMARY KEY,
  spassword VARCHAR (12) NOT NULL,
  sname CHAR (4) NOT NULL,
  gender CHAR,
  birthday DATE,
  email VARCHAR(28) NOT NULL
```

```
);
```

如果某列既没有指定 NOT NULL 选项，也没有指定 DEFAULT 子句，则可以认为该列具有默认值 NULL。

【例 7-14】 创建教师（teacher）表，其具体要求如表 7-7 所示。代码如下：

表 7-7　教师（teacher）表

列　　名	类　　型	说　　明
tn	CHAR(4)	教师号（用户名），主键
tpassword	VARCHAR(12)	口令，不能取 NULL 值
tname	CHAR(4)	教师姓名，不能取 NULL 值
dept	VARCHAR(10)	所属部门
admin	CHAR	是否管理员（是或否）

```
CREATE TABLE teacher (
   tn CHAR (4) PRIMARY KEY,
   tpassword VARCHAR (12) NOT NULL,
   tname CHAR (4) NOT NULL,
   dept VARCHAR (10) ,
   admin CHAR NOT NULL DEFAULT '否'
);
```

【例 7-15】 创建课程（course）表，其具体要求如表 7-8 所示。这里，outline 列存放课程大纲文件的扩展名，而课程号作为课程大纲文件的基本名。代码如下：

表 7-8　课程（course）表

列　　名	类　　型	说　　明
cn	CHAR(10)	课程号，主键
cname	VARCHAR(20)	课程名，不能取 NULL 值
description	VARCHAR(1000)	课程描述
credit	TINYINT	学分，不能取 NULL 值
outline	VARCHAR(5)	课程大纲文件名的扩展名
tn	CHAR(4)	负责教师的教师号，外键

```
CREATE TABLE course (
   cn CHAR (10) PRIMARY KEY,
   cname VARCHAR(20) NOT NULL,
   description VARCHAR(1000),
   credit TINYINT NOT NULL,
   outline VARCHAR(5),
   tn CHAR(4),
   CONSTRAINT fk_tn1 foreign key (tn) references teacher (tn)
);
```

【例 7-16】 创建开课（opencourse）表，用以保存某学期的开课信息，其具体要求如表 7-9 所示。其中学期号 term 的取值格式类似如“2015-2016-1”，表示 2015-2016 学年第一学期。代码如下：

表 7-9　开课（opencourse）表

列　　名	类　　型	说　　明
lid	INT	开课号，主键
term	CHAR(11)	学期号，不能取 NULL 值
cn	CHAR(10)	课程号，不能取 NULL 值，外键
tn	CHAR(4)	任课教师号，不能取 NULL 值，外键
status	CHAR	状态，不能取 NULL 值 1：选课，2：教学，3：结课 默认值：1
唯一键：term+cn+tn		

```
CREATE TABLE opencourse (
   lid INT AUTO_INCREMENT PRIMARY KEY,
   term CHAR（11）NOT NULL,
   cn CHAR(10),
   tn CHAR(4),
   status CHAR NOT NULL DEFAULT 1,
   UNIQUE uk (term, cn, tn),
   CONSTRAINT fk_cn FOREIGN KEY (cn) REFERENCES course (cn),
   CONSTRAINT fk_tn2 FOREIGN KEY (tn) REFERENCES teacher (tn)
);
```

【例 7-17】 创建学生选课（elective）表，用以保存学生选课的信息，其具体要求如表 7-10 所示。代码如下：

表 7-10　学生选课（elective）表

列　　名	类　　型	说　　明
sn	CHAR(12)	学号，不能取 NULL 值，外键
lid	INT	开课号，不能取 NULL 值，外键
score	DECIMAL(5, 2)	成绩
主键：sn+lid		

```
CREATE TABLE elective (
   sn CHAR(12),
   lid INT,
   score DECIMAL(5,2),
   PRIMARY KEY (sn, lid),
   CONSTRAINT fk_sn FOREIGN KEY (sn) REFERENCES student (sn),
   CONSTRAINT fk_lid FOREIGN KEY (lid) REFERENCES opencourse (lid)
```

```
);
```

7.5.3 显示表列表和表结构

这里介绍显示表列表、表结构和表索引等的语句。

1. 显示表列表

可以用 SHOW TABLES 语句显示指定数据库中所有表的表名列表。

```
SHOW TABLES [{FROM | IN} <db_name>]
```

如果没有指定数据库（缺省 FROM 或 IN 子句），语句显示当前数据库中所有表的表名列表。

2. 显示表结构

可以用 SHOW COLUMNS 语句显示指定表的基本结构，语法格式如下：

```
SHOW [FULL] COLUMNS {FROM | IN} <tb_name> [{FROM | IN} <db_name>]
```

语句显示指定数据库中指定表的结构，即各列的列名、数据类型、NULL 设置、默认值以及索引等信息。如果指定 FULL 选项，语句还会显示排序规则、操作权限等信息。

如果没有指定数据库（缺省{FROM | IN} <db_name>子句），语句显示当前数据库中指定表的表结构。

3. 显示表索引

可以用 SHOW INDEX 语句显示指定表的索引信息，语法格式如下：

```
SHOW INDEXE {FROM | IN} <tb_name> [{FROM | IN} <db_name>]
```

语句显示指定数据库中指定表的索引信息。如果没有指定数据库（缺省{FROM | IN} <db_name>子句），语句显示当前数据库中指定表的索引信息。

比较显示表结构 SHOW COLUMNS 语句和显示表索引 SHOW INDEX 语句，前者显示所有的列，后者只显示建有索引的列。对于多列非主键索引，前者只能显示其中第 1 列的索引信息，后者会显示所有列的索引信息，并能给出索引名以及各列在索引中的位置等信息。

4. 显示创建表的语句

语句 SHOW CREATE TABLE 显示用于创建指定表的 CREATE TABLE 语句，其语法格式如下：

```
SHOW CREATE TABLE <tb_name>
```

因为写语句时，有些选项可以不写（MySQL 会采用默认的设置），有些关键字可这样写也可那样写，所以该语句显示的 CREATE TABLE 语句与在创建表时实际使用的 CREATE TABLE 语句相比可能会有形式上的差异。

7.5.4 修改表

修改表是指修改表结构，包括修改表名，修改、添加或删除列，添加或删除索引等功

能。修改表结构的语句是 ALTER TABLE，其基本格式如下：

```
ALTER TABLE <tb_name> [<alter_specification> [,<alter_specification>]*]
```

修改规格（alter_specification）的常用格式如下：

```
{
  RENAME [TO | AS] <new_tbl_name>
  | CHANGE [COLUMN] <old_col_name> <new_col_name> <column_definition>
     [FIRST | AFTER <col_name>]
  | ALTER [COLUMN] <col_name> {SET DEFAULT <literal>| DROP DEFAULT}
  | ADD [COLUMN] <col_name> <column_definition>
     [FIRST | AFTER <col_name>]
  | ADD PRIMARY KEY (<col_name> [,<col_name>]*)
  | ADD UNIQUE [INDEX | KEY] [<index_name>] (<col_name> [,<col_name>]*)
  | ADD {INDEX | KEY} [<index_name>] (<col_name> [,<col_name>]*)
  | ADD CONSTRAINT <constraint_name>
     FOREIGN KEY (<col_name> [,<col_name>]*)
     REFERENCES [<db.name>.]<tbl_name> (<r_col_name> [,<r_col_name>]*)
        [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
        [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
  | DROP [COLUMN] <col_name>
  | DROP PRIMARY KEY
  | DROP {INDEX | KEY} <index_name>
  | DROP FOREIGN KEY <fk_constraint_name>
}
```

下面对修改规格的每个子句依次进行简单说明。

（1）RENAME 子句可以修改表名。

（2）CHANGE 子句用于修改指定列的列名，并可以对该列的有关属性或约束进行重新定义。其中，列定义（column_definition）的格式与 CREATE TABLE 语句中的列定义格式一样，包括类型及其他的属性和约束。

短语 FIRST|AFTER <col_name>指定修改后列的位置，若选择 FIRST，则该列处于表的第 1 列；若选择 AFTER col_name，则该列处于指定列的后面。

（3）ALTER 子句可以为指定列设置或重新设置（SET DEFAULT）默认值，也可以删除（DROP DEFAULT）指定列的默认值。

（4）ADD 子句添加一列，并可以指定添加的列在表中的位置。

（5）ADD PRIMARY KEY 子句用于为表创建主索引。

（6）ADD UNIQUE 子句可以为表添加一个唯一索引。

（7）ADD INDEX 或 ADD KEY 子句可以为表添加一个非唯一索引。

（8）ADD…FOREIGN KEY…REFERENCES 子句可以为表指定一个外键及相应的数据参照完整性约束，同时在外键上创建一个非唯一索引。

（9）DROP 子句可以删除指定的列。

（10）DROP PRIMARY KEY 子句可以删除表的主索引。

（11）DROP INDEX 或 DROP KEY 子句用于从表中删除指定的唯一索引或非唯一索引。

（12）DROP FOREIGN KEY 子句可以删除指定的外键定义、取消相应的数据参照完整性约束。

7.5.5 删除表

可以使用 DROP TABLE 语句删除表，其语法格式如下：

```
DROP TABLE <tb_name> [,<tb_name>]*
```

语句删除当前数据库中指定的表。需要特别注意，语句永久性地删除整个表，包括表的结构及表中所有的数据。

如果表之间存在数据参照完整性约束，那么在子表被删除之前，父表不能被删除。如果在一条 DROP TABLE 语句中同时删除父表和子表，那么子表应列在父表之前。

7.6 数据的插入、更新和删除

定义好了表结构，就可以向表中插入（添加）数据行，需要时还可以更新（修改）表中的数据，或者删除表中的数据行。

7.6.1 插入数据

插入数据行的 SQL 语句是 INSERT。插入的方式包括插入完整的行、插入行的一部分、插入多行、插入子查询的结果等，下面分别介绍这些方式。

1. 插入完整的行

要用 INSERT 语句向表插入完整的一行，可以使用下面格式：

```
INSERT [INTO] [<db_name>.]<tb-name>
  VALUES ({<expr>| DEFAULT} [, {<expr>| DEFAULT}]*)
```

除了指定表名，VALUES 后面的括号内要给出插入行在各列的取值。MySQL 会把其中的第 1 个值保存在新行的第 1 列，第 2 个值保存在新行的第 2 列，依次类推。也就是说，使用此格式时，需要清楚表中各列的次序，VALUES 后面括号内值的数目要与表中列的数目相等，且一一对应。

给每一列指定的值可以是表达式，也可以是 NULL 或者 DEFAULT。如果指定为 NULL，那么该列应该允许取 NULL 值。如果指定为 DEFAULT，那么该列应该定义了默认值或者允许取 NULL 值。

【例 7-18】 在教师表 teacher 中插入一行，所属部门为'信息学院'，是否管理员取默认值，这里为'否'值。假定当前数据库为 elective_manage。代码如下：

```
INSERT INTO teacher VALUES('1011', '333333', '李国柱', '信息学院', DEFAULT);
```

2. 插入行的一部分

要用 INSERT 语句向表插入一行的一部分，可以使用下面格式：

```
INSERT [INTO] [<db_name>.]<tb-name> (<col_name> [,<col_name>]*)
  VALUES ({<expr>| DEFAULT} [, {<expr>| DEFAULT}]*)
```

该格式中，除了在 VALUES 后面的括号内给出值列表，还需要在表名后的括号内给出列名列表。MySQL 会把值列表中第 1 个值保存在列名列表中的第 1 列，把值列表中第 2 个值保存在列名列表中的第 2 列，依次类推。这里，列名列表中各列的次序并不要求与表中各列的实际次序一致。

严格来说，该格式也是在表中插入完整的一行，只是语句只为其中的一些列指定了值，而让其他列取默认值或 NULL 值。要使用此格式，表中没有在列名列表中列出的列必须满足：可以取 NULL 值或者定义了默认值。如果定义了默认值，插入的新行在该列取默认值，否则取 NULL 值。

【例 7-19】 在课程表 course 中插入一行。没有为课程描述和课程大纲指定值，由于这两列都没有定义默认值，所以它们取 NULL 值。假定当前数据库为 elective_manage。代码如下：

```
INSERT INTO course (cn, cname, credit, tn) VALUES ('0901011001', '会计学原
理', 3, '1011');
```

要为所有列指定值也可以使用此格式，即在列名列表中列出数据表中所有的列。这种做法虽然有些烦琐，但有其好处：一是值与列名一一对应，不容易出现次序上的错误；二是当数据表中各列的次序发生变化时，语句代码仍然是有效的。

3. 插入多行

要用 INSERT 语句一次插入多行，可以使用下面格式：

```
INSERT [INTO] [<db_name>.]<tb-name> [(<col_name> [,<col_name>]*)]
  VALUES ({<expr>| DEFAULT} [, {<expr>| DEFAULT}]*)
         [, ({<expr>| DEFAULT} [, {<expr>| DEFAULT}]*)]*
```

为向表中插入多行，可以在关键字 VALUES 后跟多对括号，两对括号之间用逗号分隔，每对括号内指定一行数据。

【例 7-20】 在学生表 student 中插入 3 行。代码如下：

```
INSERT INTO elective_manage.student (sn, spassword, sname, birthday, gender,
email)
  VALUES ('201209031001', '222222', '胡文海', '1988-7-12', '男', 'hwh@163.com'),
         ('201209031002', '222222', '赵立刚', '1988-11-12', '男',
         'zhaolg@163.com'),
          ('201209031003', '222222', '李红霞', '1989-10-10', '女',
         'lhx@cueb.edu.cn');
```

使用此格式时，可以给出列名列表，也可以缺省列名列表。如果给出列名列表，那么应该给列出的各列按顺序指定值。如果缺省列名列表，那么应该给所有的列按它们在数据表中的顺序指定值。

4. 插入子查询的结果

可以把一个子查询的查询结果作为行的数据插入表中，其语法格式如下：

```
INSERT [INTO] [<db_name>.]<tb_name> [(<col_name> [,<col_name>]*)]
   SELECT <select_expr> [,<select_expr>]* ...
```

语句把 SELECT 子查询的结果插入到目标表（tb_name）中。SELECT 语句的语法及用法会在本书 7.7 节详细介绍，这里只需关注该子查询的结果。如果在目标表名后指定了列名列表，那么查询结果（select_expr）的列数就要与列名（col_name）列表中的列数相等。如果在目标表名后缺省列名列表，那么查询结果（select_expr）的列数就要与数据表中的实际列数相等。查询结果中的各列值会按顺序保存到目标表对应的列中。查询结果有多少行就会在目标表中插入多少行。

7.6.2 更新数据

可以使用 SQL UPDATE 语句来更新表中数据，其基本语法格式如下：

```
UPDATE [<db_name>.]<tb_name>
   SET <col_name>={<expr>| DEFAULT} [,<col_name>={<expr>| DEFAULT}]*
   [WHERE <condition>]
```

SET 子句指定为一列或多列设置新的值或默认值。WHERE 子句指定更新哪些数据行，语句只更新满足条件（condition）的数据行。如果缺省 WHERE 子句，MySQL 将更新表中所有数据行，这通常不是应用所希望的。

【例 7-21】 更新学生表中学号为'201209031001'的学生数据：将登录口令改为'333333'、出生日期改为'1989-1-11'。代码如下：

```
UPDATE elective_manage.student
   SET spassword = '333333', birthday = '1989-1-11'
   WHERE sn = '201209031001';
```

7.6.3 删除数据

删除数据是指删除表中的某些行。可以使用 SQL DELETE 语句来删除数据，其基本语法格式如下：

```
DELETE FROM [<db_name>.]<tb_name> [WHERE <condition>]
```

WHERE 子句指定删除哪些数据行，语句删除满足条件（condition）的数据行。如果缺省 WHERE 子句，MySQL 将删除表中所有数据行。

【例 7-22】 从学生表中删除学号为'201209031001'的学生数据。代码如下：

```
DELETE FROM elective_manage.student WHERE sn = '201209031001';
```

除非确实要删除表中所有数据行，否则总是使用带 WHERE 子句的 DELETE 语句。在执行 DELETE 语句之前，可以先使用带相同 WHERE 子句的 SELECT 语句，看看查询结果

是否确实是要删除的数据。

7.7 查　　询

毫无疑问，查询数据是用户最为频繁使用的操作。可以使用 SQL SELECT 语句实现数据查询。SELECT 语句的语法成分比较复杂，功能比较强大。下面首先对 SELECT 语句的基本格式及功能作一个初步介绍，然后再分专题逐步介绍语句的使用。

7.7.1 SELECT 语句

下面是 MySQL 的 SELECT 语句的基本格式：

```
SELECT [DISTINCT ]
  {*|<select_expr> [[AS] <col_alias>] [,<select_expr> [[AS] <col_alias>]]*}
  FROM <tbl_name> [[AS] <tbl_alias>] [,<tbl_name> [[AS] <tbl_alias>]]*
  [WHERE <condition>]
  [GROUP BY {<col_name>|<expr>|<position>}
          [,{<col_name>|<expr>|<position>}]*
  [HAVING <condition>]
  [ORDER BY {<col_name>|<expr>|<position>} [ASC | DESC]
          [,{<col_name>|<expr>|<position>} [ASC | DESC]]*
  ]
  [LIMIT [<offset>,] <row_count>]
```

该格式主要包括 SELECT、FROM、WHERE、GROUP BY、HAVING 和 ORDER BY 等子句，其中 SELECT 和 FROM 子句是必选的，其他子句是可选的。各子句的作用如下：

（1）SELECT 子句指定要查询的数据，通常是表中的某些列或是基于列的某些表达式。

（2）FROM 子句指定要查询的数据来自哪些表。后面先从单表查询开始介绍，多表查询主要在连接查询中介绍。

（3）WHERE 子句指定查询条件，即从表中选择哪些行。

（4）GROUP BY 子句说明如何对数据行进行分组，进而实现分组汇总等功能。

（5）HAVING 子句必须与 GROUP BY 子句配合使用，用于指定分组查询的条件，即哪些分组汇总后满足查询条件。

（6）ORDER BY 子句用于对查询结果进行排序。

7.7.2 指定列

这里讨论 SELECT 子句的用法。SELECT 子句指定要查询表中哪些列的数据，同时也指定了查询结果包含哪些列。

1. 指定单列、多列和所有列

要查询单列数据，只需在关键字 SELECT 后给出相应的列名即可。

【例 7-23】 从学生表中查询学生姓名。代码如下：

```
SELECT sname FROM student;
```

要查询多列数据，可以在关键字 SELECT 后给出所需的各列列名，各列名之间用逗号（,）分隔。

【例 7-24】 从学生表中查询学号、姓名、性别和出生日期信息。代码如下：

```
SELECT sn, sname, gender, birthday FROM student;
```

要查询表中所有列的数据，一般不需要列出所有列的列名，只要在关键字 SELECT 后放置一个星号（*）通配符。例如，下面语句查询学生的各项信息：

```
SELECT * FROM student;
```

此时，MySQL 将获取表中所有列，查询结果中列的顺序通常与列在表中的顺序是一致的。

2. 去除重复行

在指定单列或多列的查询中，查询结果往往会包含一些重复的行，即这些行在各列的取值都相同。可以使用可选关键字 DISTINCT 去除重复行，它能确保查询返回的结果中，各行都是不一样的。

【例 7-25】 查询学生的姓名和性别信息。代码如下：

```
SELECT DISTINCT sname, gender FROM student;
```

由于指定了关键字 DISTINCT，所以如果出现两位或更多的学生，他们的姓名和性别都相同，那么查询结果中只保留一行。

3. 创建计算列

查询结果中各列的数据不见得要与表中相应列中的数据完全相同，有时候查询需要对表中数据进行计算、转换或格式化再返回。就如语句的语法格式所表示的，SELECT 子句指定的不一定是列名，也可以是普通的表达式。这里，把这种普通的表达式（而非简单的列名）称为计算列。

在指定计算列时，可以使用 MySQL 的各种运算符、函数等。比如，对数值型数据，可能会进行算术运算。在 MySQL 中，算术运算符包括+（加）、–（减）、*（乘）、/（除）、%（求余）等。对字符串，可能需要进行子串提取、连接等运算。

【例 7-26】 从开课表中查询学期、课程号和教师号 3 项信息，其中第 1 列是计算列。代码如下：

```
SELECT CONCAT(SUBSTR(term, 1, 9), '学年第', RIGHT(term, 1), '学期'), cn, tn
  FROM opencourse;
```

这里，学期（term）列的类型是 CHAR(11)，其值的格式类似“2015-2016-1”，实际含义是“2015-2016 学年第 1 学期”。语句通过计算列显示学期的实际含义。

该计算列用到了 MySQL 的 3 个字符串函数，SUBSTR 和 RIGHT 函数用于获取子串，CONCAT 函数能够实现字符串的连接。

4. 使用列别名

默认情况下，查询结果中各列的列名就是在 SELECT 子句中指定的列名或表达式。可

以使用可选项[AS] <col_alias>为某列或计算列指定别名，这个别名将成为查询结果中相应列的列名。

【例 7-27】 为例 7-26 语句中的计算列指定别名，使查询结果更加易于阅读。代码如下：

```
SELECT CONCAT(SUBSTR(term, 1, 9),'学年第',RIGHT(term, 1),'学期') AS 学期,cn,tn
  FROM opencourse;
```

列别名可用于 GROUP BY、HAVING 和 ORDER BY 子句中。当 PHP 等应用需要处理查询结果时，为计算列指定别名是非常有必要的。

7.7.3 选择行

通常，查询操作总是从表中选择所需要的行返回，而不会返回所有行。选择行的任务由 WHERE 子句完成，WHERE 子句通常放置在 FROM 子句之后。

WHERE 子句中的查询条件（condition）是一个表达式。在执行 SELECT 语句时，对表中的每一行，如果该条件表达式的计算结果为 TRUE，那么就说条件成立，该行将被选中返回。

实际上，MySQL 并不支持严格意义上的布尔型数据。其文字 TRUE 和 FALSE 分别代表数值 1 和 0。所以在这里，所谓条件成立是指条件表达式的计算结果是一个非零的数值。条件表达式可以使用 MySQL 的各种运算符，也可以使用除聚集函数之外的任何函数。下面主要简单介绍比较运算符和逻辑运算符的使用。

1. 比较运算符

常用的比较运算符包括：

>（大于） >=（大于等于） <（小于） <=（小于等于） =（等于） !=（不等于）

比较运算符可以对各种类型数据进行比较，需要时会自动对类型进行转换。比较运算符的运算结果总是 1（TRUE）、0（FALSE）或 NULL。

【例 7-28】 从开课表中查询 2015-2016 学年第 1 学期的开课信息。代码如下：

```
SELECT * FROM opencourse WHERE term = '2015-2016-1';
```

2. 比较 NULL 值

在用上面比较运算符构建条件时，只要有一个操作数为 NULL 值，运算结果就是 NULL 值。比如条件'ming'!= NULL 和 NULL=NULL 的运算结果均为 NULL，条件不成立。

为了有效比较 NULL 值，可以使用 IS [NOT] NULL 短语，它可以测试列或表达式的值是否为 NULL 值。

【例 7-29】 从课程表中查询课程大纲（outline）为 NULL 的课程。代码如下：

```
SELECT * FROM course WHERE outline IS NULL;
```

3. 逻辑运算符

常用的逻辑运算符包括（按优先级从高到低）：

NOT（逻辑非） AND（逻辑与） XOR（逻辑异或） OR（逻辑或）

逻辑运算符的作用是对已有的条件再做进一步的操作。当用 AND 将两个条件连接起来

时，只有两个条件都成立时，整个条件才会成立。当用 OR 将两个条件连接起来时，只要有一个条件成立，整个条件就成立。当用 XOR 将两个条件连接起来时，有且只有一个条件成立，整个条件才成立。运算符 NOT 是单目运算符，其功能是否定跟在它之后的条件。

与关系运算符一样，逻辑运算符的运算结果总是 1（TRUE）、0（FALSE）或 NULL。

【例 7-30】 从开课表中查询 2015-2016 学年第 1 学期的且课程号为'0601011001'或者教师号为'1100'的开课信息。代码如下：

```
SELECT * FROM opencourse
  WHERE term='2015-2016-1' AND (cn='0901011001' OR tn='1100');
```

在条件表达式中，运算符 AND 的优先级比运算符 OR 的优先级高，所以该例中的括号是必要的，它改变了运算次序。

7.7.4 使用谓词

使用谓词可以更好或更方便地构建查询条件。这里介绍 IN、BETWEEN…AND、LIKE 这 3 个谓词。

1. 谓词 IN

使用谓词 IN 可以测试列或表达式的值是否为值列表中的一个，其语法格式如下：

```
<expr> [NOT] IN (<expr1> [,<expr2>]*)
```

这里，<expr>通常是列名，值列表放置在 IN 后面的一对括号内。功能上，这个 IN 条件表达式与下面使用运算符 OR 的条件表达式是相同的：

```
[NOT] (<expr>=<expr1> [OR <expr>=<expr2>]*)
```

【例 7-31】 从开课表中查询教师号为'1100'、'1090'或'1019'的开课信息。代码如下：

```
SELECT * FROM opencourse WHERE tn IN ('1100', '1090', '1019');
```

2. 谓词 BETWEEN…AND

使用该谓词可以测试列或表达式的值是否落在指定的区间，其语法格式如下：

```
<expr> [NOT] BETWEEN <expr1> AND <expr2>
```

这里，<expr>通常是列名，<expr1>和<expr2>分别表示区间的下限值和上限值。功能上，这个条件表达式与下面使用运算符 AND 的条件表达式是相同的：

```
[NOT] (<expr>>=<expr1> AND <expr><=<expr2>)
```

【例 7-32】 从学生表中查询 1988-09-01 至 1989-08-31 期间出生的学生信息。代码如下：

```
SELECT *
  FROM student
  WHERE birthday BETWEEN '1988-09-01' AND '1989-08-31';
```

3. 谓词 LIKE

使用谓词 LIKE 可以实现对字符串的通配比较，其语法格式如下：

```
<expr> [NOT] LIKE <pattern>
```

这里，<expr>通常是列名，<pattern>通常是包含通配符的字符串文字。可以使用的通配符有两个：%（百分号）代表任意的字符序列（包括零个字符），_（下画线）代表任意单个字符。

【例 7-33】 从课程表中查询课程名含“工程”字样的课程信息。代码如下：

```
SELECT * FROM course WHERE cname LIKE '%工程%';
```

7.7.5 排序查询结果

通过在 SELECT 子句中指定列，可以规定查询结果中列的次序。通过使用 ORDER BY 子句，可以规定查询结果中行的次序。

1. ORDER BY 子句

ORDER BY 子句用于对查询结果各行按指定列的值进行排序。ORDER BY 子句指定的排序依据可以是列名（或表达式），也可以是列别名或者列的序号；可以是在 SELECT 子句中指定的列（或表达式），也可以是其他的列（或表达式）。

【例 7-34】 从学生表中查询男同学各项信息，查询结果按出生日期升序排序。代码如下：

```
SELECT * FROM student WHERE gender='男' ORDER BY birthday;
```

ORDER BY 子句指定的排序依据可以是单列（或表达式），也可以是多列（或表达式）。如果是多列（或表达式），两列（或表达式）之间用逗号分隔。

当 ORDER BY 子句指定多列（或表达式）时，MySQL 先按前面的列（或表达式）的值对各行进行排序，对其值相同的行，再按后面的列（或表达式）的值进行排序。

默认情况下，查询结果中的各行是按指定列（或表达式）的值进行从小到大排序的，即升序排序。如果要从大到小排序，即降序排序，可以在相应列（或表达式）后面指定关键字 DESC。如果缺省或者指定 ASC，则表示要升序排序。ORDER BY 子句中列出的每个列（或表达式）后面都可以指定 ASC 或 DESC。

【例 7-35】 从学生选课表中查询包括成绩在内的各项信息，查询结果按成绩降序排序，如果成绩相同按学号升序排序。代码如下：

```
SELECT * FROM elective ORDER BY score DESC, sn ASC;
```

语句中最后的关键字 ASC 可以省略。

2. LIMIT 子句

可以使用 LIMIT 子句限制查询结果的返回行数，例如：

```
LIMIT 5              # 返回最前面的 5 行
LIMIT 5, 10          # 返回从第 5 行开始的 10 行
```

这里的行号是从 0 开始计算的，即第一行的行号为 0。若 LIMIT 后面跟 1 个数字，表示从第 0 行开始的若干行。若 LIMIT 后面跟 2 个数字，那么第 1 个数字表示起始行号，第 2 个数字表示行数。

7.7.6 分组汇总

这里首先介绍如何利用聚集函数对表中数据进行汇总，然后再介绍如何对表中数据进行分组并对每一组分别进行汇总。

1．聚集函数

在 SELECT 子句中指定列时也可以使用聚集函数，这时语句返回的查询结果将会是汇总数据，而不是表中的原始数据。表 7-11 列出一些常用的聚集函数。

表 7-11　SQL 常用聚集函数

聚集函数	说　明
COUNT({* | [DISTINCT] <expr>})	返回行数
MAX(<expr>)	返回指定列或表达式（<expr>）的最大值
MIN(<expr>)	返回指定列或表达式（<expr>）的最小值
AVG([DISTINCT] <expr>)	返回指定列或表达式（<expr>）的平均值
SUM([DISTINCT] <expr>)	返回指定列或表达式（<expr>）的和

其中 COUNT(*)不涉及具体的列，它会返回总的行数。COUNT([DISTINCT] <expr>)和其他聚集函数在汇总时都会忽略 NULL 值。如果指定可选项 DISTINCT，那么在汇总时还会忽略重复值。

【例 7-36】 假设学生选课表包含以下数据：

```
-----------------------------------
 sn             lid  score
-----------------------------------
 201208031001    1  80.00
 201208031001    2  85.00
 201208031001    3  92.00
 201208031002    2  72.00
 201208031002    3  85.00
 201208031002    4  NULL
 201209031003    2  85.00
 201209031003    4  NULL
-----------------------------------
```

请使用聚集函数 COUNT 和 AVG 对成绩列（score）进行计数和求平均值。代码如下：

```
SELECT COUNT(*) c1, COUNT(score) c2, COUNT(DISTINCT score) c3,
       AVG(score) avg1, AVG(DISTINCT score) avg2
  FROM elective;
```

下面是语句执行的输出结果：

```
-----------------------------------------------
 c1  c2  c3  avg1       avg2
-----------------------------------------------
 8   6   4   83.166667  82.250000
-----------------------------------------------
```

2. GROUP BY 子句

实现分组汇总查询的方法是：用 GROUP BY 子句对查询出来的各行进行分组，利用聚集函数对每一组数据分别进行汇总和计算。

GROUP BY 子句指定的分组依据可以是列名（或表达式），也可以是列别名或者列的序号；可以是在 SELECT 子句中指定的列（或表达式），也可以是其他的列（或表达式）。

GROUP BY 子句指定的分组依据可以是单列（或表达式），也可以是多列（或表达式）。如果是多列（或表达式），各列（或表达式）之间用逗号分隔。

SELECT 语句执行时，MySQL 会把所有在 GROUP BY 子句指定的各列（或表达式）上都取相同值的行分为一组。

3. HAVING 子句

HAVING 子句总是与 GROUP BY 子句配合使用，用于指定分组查询的条件，即哪些分组汇总后满足查询条件。

【例 7-37】对学生选课表中的成绩（score）进行分组求平均值，分组依据是开课号（lid），分组查询条件是该组至少有一个成绩，查询结果按平均成绩降序排序。代码如下：

```
SELECT lid, AVG(score)
  FROM elective
  GROUP BY lid HAVING COUNT(score)>0
  ORDER BY 2 DESC;
```

当 SELECT 语句既包含 WHERE 子句，又包含 GROUP BY、HAVING 和 ORDER BY 子句时，首先用 WHERE 子句过滤查询内容，然后基于 GROUP BY 子句对过滤后的内容进行聚集计算并用 HAVING 子句过滤聚集计算后的内容，最后用 ORDER BY 子句对查询结果进行排序并返回。

7.7.7 使用子查询

子查询是一个嵌套在其他语句中的 SELECT 语句。子查询 SELECT 语句需要放置在一对括号内。经常地，子查询嵌套在另外一个 SELECT 语句中，这里可以把外面的 SELECT 语句称为外查询。一个子查询也可以包含自己的子查询，嵌套的子查询的数目没有限制。

子查询的查询结果可以是单个值、单列、单行或表（多行多列），但这要受它所处上下文的限制。即有时子查询的查询结果只能是单个值或单列，有时则可以是单行或表等。下面介绍子查询的几种常用形式。

1. 利用子查询构建查询条件

最典型的，子查询可以作为比较运算符的一个操作数。一般的格式如下：

```
<expr> <comparison_operator> (<subquery>)
```

其中，<expr>是另一个操作数，通常是某个列名；<comparison_operator>是各种比较运算符。此时，子查询的结果应该是单个值。

【例 7-38】 利用子查询从学生表中查询与学号为'201209031001'的学生在同一天出生所有学生的信息。代码如下：

```
SELECT *
  FROM student
  WHERE birthday = (SELECT birthday
                    FROM student
                    WHERE sn = '201209031001');
```

2. ALL 或 ANY 子查询

这里介绍量词 ALL 和 ANY。它们通常与子查询配合使用，一般格式如下：

```
<expr> <comparison_operator> {ALL | ANY} (<subquery>)
```

这里，子查询的结果应该是单列。如果选用 ALL，那么只有<expr>的值与子查询结果中所有值都符合比较要求，条件才算成立，否则条件不成立。如果选用 ANY，那么只要<expr>的值与子查询结果中某个值符合比较要求，条件就算成立，否则条件不成立。

【例 7-39】 利用子查询从学生选课表（elective）中选取满足下面条件的行：其成绩比开课号（lid）为 2 的行的所有成绩都高。代码如下：

```
SELECT *
  FROM elective
  WHERE score > ALL (SELECT score
                     FROM elective
                     WHERE lid=2);
```

功能上等价于

```
SELECT *
  FROM elective
  WHERE score > (SELECT MAX(score)
                 FROM elective
                 WHERE lid=2);
```

3. IN 子查询

前面介绍过谓词 IN 的使用。这里介绍谓词 IN 与子查询配合使用情况，一般格式如下：

```
<expr> [NOT] IN (<subquery>)
```

这里，子查询的结果应该是单列。整个表达式测试<expr>的值是否等于子查询结果中的某个值。

4. EXISTS 子查询

EXISTS 是谓词，总是和子查询配合使用，一般格式如下：

```
[NOT] EXISTS (<subquery>)
```

该条件测试子查询结果是否为空（没有返回行）。如果没选 NOT，那么当查询结果不为空时，条件成立，否则条件不成立。如果指定 NOT，那么测试结果正好相反。

【例 7-40】 查询到目前为止在开课表中还没有开课信息的教师。代码如下：

```
SELECT tn, tname
  FROM teacher
  WHERE NOT EXISTS (SELECT *
                      FROM opencourse
                      WHERE opencourse.tn = teacher.tn);
```

这里，子查询的 WHERE 子句使用了完全限定列名，即用表名限定列名。因为它要做的比较是：opencourse 表中当前行的 tn 列值是否等于 teacher 表中当前行的 tn 列值。由于对于子查询来说，当前表是 opencourse，所以 opencourse.tn 可以简写为 tn，但 teacher.tn 则一定不能省略表名，否则就变成 opencourse 表中的 tn 列与自身进行比较了。

上面语句的功能等价于

```
SELECT tn, tname
  FROM teacher
  WHERE tn NOT IN (SELECT tn FROM opencourse);
```

5. 利用子查询构建计算列

子查询也可以作为一个计算列出现在 SELECT 子句中，此时，子查询的查询结果应该是单个值。

【例 7-41】 利用子查询从开课表和学生选课表中查询每位教师（tn）所开课（cn）的选课人数。代码如下：

```
SELECT tn, cn, (SELECT COUNT(*)
                 FROM elective
                 WHERE lid = opencourse.lid) AS 人数
  FROM opencourse;
```

7.7.8 连接查询

连接查询是 SQL SELECT 语句能够执行的最重要的操作之一。连接查询是基于多个表的查询，涉及的表中的数据往往存在某种关系。比如学生表和学生选课表，选课表中的任何一条选课信息一定是属于某个学生的。我们可以在这两个表上进行连接查询，了解学生的选课情况。在这种连接查询中，定义表间数据的关系，或者说创建表间的连接条件，是至关重要的。

1. 创建连接条件

连接条件一般是通过外键来创建的。一个表的外键通常是另一个表的主键，两个表的连接条件一般就是外键的值要与主键的值相等。通常，只有把满足这样条件的两行数据连接在一起才是有意义的。

【例 7-42】 查询 2015—2016 年度第 1 学期的开课信息，查询结果包括开课号、课程名称和任课教师姓名 3 项信息。代码如下：

```
SELECT lid, course.cname, teacher.tname
  FROM opencourse, course, teacher
  WHERE opencourse.cn = course.cn AND
        opencourse.tn = teacher.tn AND
        term = '2015-2016-1';
```

由于开课信息主要在开课表中，而课程名称只有课程表中有，教师姓名只有教师表中有，所以该例涉及 3 个表的连接查询。WHERE 子句中，前面两个条件称为连接条件，而最后一个一般称为过滤条件。

2. 表别名与自连接

需要时，可以在 FROM 子句中为表指定别名。这样，在其他子句中需要引用某个表时，就可以用别名代替表名。

表别名有时可使整个语句简单明了，但通常不是必需的。而在自连接查询中，表别名就是必不可少的了。所谓自连接是指将一个表与其自身进行连接。

【例 7-43】 利用连接查询从学生表中查询与学号为'201209031001'的学生在同一天出生所有学生的信息。代码如下：

```
SELECT s1.*
  FROM student s1, student s2
  WHERE s1.birthday = s2.birthday AND s2.sn = '201209031001';
```

自连接查询中，连接条件往往不是建立在主键值相等上。如本例中，连接条件建立在 birthday 列值相等上，另外需要表 s2 的 sn 值为'201209031001'。

3. JOIN…ON 短语

JOIN…ON 短语是 FROM 子句的一部分，使用它可以使 FROM 子句在指定要连接的表的同时指定连接条件。

例 7-42 中的语句也可以用 JOIN…ON 短语改写成如下：

```
SELECT lid, course.cname, teacher.tname
  FROM opencourse
    JOIN course ON opencourse.cn = course.cn
    JOIN teacher ON opencourse.tn = teacher.tn
  WHERE term = '2015-2016-1';
```

4. 外连接

前面看到的连接查询都是内连接，也叫普通连接。内连接仅把两个表中满足连接条件的行连接在一起返回。相应的，还有外连接。外连接是指将那些不满足连接条件的行也包含在查询结果中。外连接分为左连接和右连接。

左连接使用 LEFT [OUTER] JOIN 关键字，其查询结果除了包含内连接的结果，还包括左边表中不满足连接条件的行，此时属于右边表的各列取 NULL 值。

右连接使用 RIGHT [OUTER] JOIN 关键字，其查询结果除了包含内连接的结果，还包括右边表中不满足连接条件的行，此时属于左边表的各列取 NULL 值。

【例 7-44】 利用连接查询从开课表和学生选课表查询每位教师（tn）所开课（cn）的选课人数。代码如下：

```
SELECT tn, cn, COUNT(sn) 人数
  FROM opencourse LEFT JOIN elective ON opencourse.lid = elective.lid
  GROUP BY opencourse.lid;
```

这里采用左连接，并在此基础上进行分组汇总。之所以采用左连接是要确保那些目前还没有学生选的开课信息在查询结果中仍然存在，只不过它们的选课人数为零。

习　题　7

1. 根据要求写 MySQL SQL 语句

（1）在 MySQL 监控程序命令行状态下，创建一个账户，该账户的用户名为'zhang'，密码是'123456'，只能从本地登录。

（2）在 MySQL 监控程序命令行状态下，授予用户账户'zhang'@'localhost'可以在数据库 elective_manage 上进行数据操纵（select、insert、update、delete）的权限。

（3）在 MySQL 监控程序命令行状态下，创建一个名为 mydb 的数据库，数据库的默认字符集为 UTF-8，默认排序规则为 utf8_bin。

（4）在 MySQL 监控程序命令行状态下，选择 mydb 为当前数据库。

（5）在 MySQL 监控程序命令行状态下，显示 MySQL 服务器上数据库名的列表。

（6）在 MySQL 监控程序命令行状态下，显示当前数据库中 student 表的表结构。

（7）在开课表（opencourse）中插入一行：课程号（cn）为'0901011000'，任课教师号（tn）为'1011'，学期（term）为'2014-2015-2'。

（8）从学生选课表（elective）中删除一行：学号（sn）为'201209031003'、开课号（lid）为 1。

（9）查询 2014-2015 学年第二学期的开课信息，查询结果包括：开课号、课程号、课程名和任课教师姓名。

（10）查询开课号（lid）为 12 的学生选课情况，查询结果包括学号、学生姓名和成绩，查询结果按学号升序排序。

2. 简答题

（1）如何利用命令行客户端程序（mysql）登录 MySQL 服务器？

（2）在 MySQL 中，怎样更改一个账户的密码？

（3）试说明 SELECT 语句中 GROUP BY、HAVING 和 ORDER BY 子句的作用。

第 8 章　PHP 访问数据库

本章主题：

- 利用 MySQLi 扩展访问数据库;
- 使用 MySQLi、MySQLi_RESULT 和 MySQLi_STMT 类;
- 利用 PDO 扩展访问数据库;
- 使用 PDO、PDOStatement 和 PDOException 类;
- 管理事务;
- 分页显示。

PHP 应用通常利用 PHP 的 MySQLi 扩展和 PDO 扩展访问数据库。MySQLi 扩展主要提供 MySQLi、MySQLi_RESULT 和 MySQLi_STMT 这 3 类，程序员可以使用这些类建立与 MySQL 数据库的连接、访问 MySQL 数据库以及处理查询结果。

PDO 是一种数据访问抽象层，它为 PHP 程序员提供了一个轻量级的一致的编程接口。利用它，PHP 应用能够用相同的代码访问各种不同的数据库。PDO 扩展提供的应用程序编程接口主要包括 PDO、PDOStatement 和 PDOException 类。

8.1　使用 MySQLi 访问数据库

MySQLi 是 PHP 的一个扩展，它提供了一个面向对象形式的应用程序编程接口 API。PHP 程序员利用该 API 可以建立与 MySQL 数据库的连接，进而访问数据库并对查询结果进行处理。

MySQLi 提供的 API 主要包括 MySQLi、MySQLi_RESULT 和 MySQLi_STMT 这 3 类。本节主要涉及前面两个类，MySQLi_STMT 类将在下一节介绍。

8.1.1　建立与 MySQL 服务器的连接

要访问 MySQL 数据库，首先要建立与 MySQL 服务器的连接。在 MySQLi 中，要建立与 MySQL 服务器的连接，就是要实例化 MySQLi 类，同时提供为连接 MySQL 服务器所需的相关参数。

1. 实例化 MySQLi

创建 MySQLi 类的一个实例对象可以实现与指定 MySQL 服务器的连接。下面是 MySQLi 类的构造方法的头，其格式如下：

```
__construct([string $host, string $username, string $passwd
   [, string $dbname = "" [, int $port = ini_get("mysqli.default_port")]]]
)
```

其中各参数的含义如下：

- $host：MySQL 服务器所在主机的域名或 IP 地址。
- $username：MySQL 服务器某账户的用户名。
- $passwd：账户的密码。
- $dbname：为后续查询操作指定默认数据库。默认值为空串。
- $port：MySQL 服务器的端口号。默认值在 php.ini 文件中设置，通常为 3306。

如果连接成功，接下来就可以调用 MySQLi 对象的相关方法对数据库进行所需的查询等操作。

例如，下面代码创建一个 MySQLi 类的实例对象，并建立与处于本地的 MySQL 服务器中的选课管理数据库 elective_manage 的连接：

```
$mysqli = new mysqli("127.0.0.1", "root", "", "elective_manage", 3306);
```

也可以分两步建立与 MySQL 服务器的连接。首先在创建 MySQLi 类的实例对象时不带任何参数，例如：

```
$mysqli = new mysqli();
```

这样创建的 MySQLi 对象（$mysqli）没有建立与某个 MySQL 服务器的连接，还无法访问数据库。接下来可以调用对象的 connect 方法建立与指定 MySQL 服务器的连接，其格式如下：

```
connect(string $host, string $username, string $passwd
   [, string $dbname="" [, int $port=ini_get("mysqli.default_port")]]
)
```

其中各参数的含义与前述构造方法中各参数的含义相同。

例如，下面代码利用已有的 MySQLi 对象（$mysqli），建立与处于本地的 MySQL 服务器的连接，并指定选课管理数据库 elective_manage 为默认数据库：

```
$mysqli->connect("127.0.0.1", "root", "", "elective_manage", 3306);
```

说明：需要通过对象及运算符->来访问特定对象的方法，例如$mysqli->connect(…)。

2. 连接错误码和错误信息

在建立与 MySQL 服务器的连接时，如果有关参数指定不正确，会导致连接失败，并发出警告信息。可以访问 MySQLi 对象的有关属性，了解连接是否成功。

```
int $connect_errno
```

该属性保存着最近一次执行与 MySQL 服务器连接操作产生的错误代码。如果连接操作成功，没有产生错误，则该属性值为 0。

```
string $connect_error
```

该属性保存着最近一次执行与 MySQL 服务器连接操作产生的错误信息。如果连接操作成功，没有产生错误，则该属性值为 NULL。

说明：需要通过对象及运算符->来访问特定对象的属性，如$mysqli->connect_errno。这里$mysqli 表示一个 MySQLi 对象，属性名（如 connect_errno）不要带$符号。

3. 关闭连接

可以调用 MySQLi 对象的 close 方法关闭与 MySQL 服务器的连接，释放相关的资源。语法格式如下：

```
bool close()
```

如果操作成功，方法返回 true；否则方法返回 false。

当 PHP 脚本代码运行结束时，当前与 MySQL 服务器的连接也会自动关闭。但明确调用该方法关闭连接，可以及时释放相关资源，是一种好的习惯。

【例 8-1】 创建与 MySQL 数据库的连接。以超级用户身份登录本地的 MySQL 服务器，并建立与 elective_manage 数据库的连接。代码如下：

```
<!DOCTYPE html>
<?php
header("Content-type:text/html;charset=UTF-8");

@ $mysqli = new mysqli("127.0.0.1", "root", "", "elective_manage", 3306);
if ($mysqli->connect_errno) {
   echo "不能连接到数据库<br/>";
   return;
}

print "成功连接至数据库！";
$mysqli->close();
?>
```

8.1.2 访问 MySQL 数据库

访问 MySQL 数据库是指向 MySQL 服务器发送并执行 SQL 语句。在访问数据库前，还需要做一些准备工作，如设置字符集、选择数据库等。

1. 设置字符集

当从服务器获取数据或向服务器发送数据时，都会涉及字符编码，即采用何种字符集。调用 MySQLi 对象的 set_charset 方法可以设置所要采用的字符集。语法格式如下：

```
bool set_charset(string $charset)
```

方法为当前连接设置默认字符集，参数$charset 指定要设置的默认字符集的名称。如果设置成功，方法返回 true；否则方法返回 false。

另外，也可以使用 SQL SET NAMES 语句为当前连接设置默认字符集。语法格式如下：

```
SET NAMES <charset_name> [COLLATE <collation_name>]
```

要获取当前连接的默认字符集，可以调用 MySQLi 对象的 character_set_name 方法。语

法格式如下：

```
string character_set_name(void)
```

2. 选择数据库

在建立与 MySQL 服务器连接时，可以指定一个默认的数据库，也称为当前数据库。但无论是否已经指定，都可以调用 MySQLi 对象的 select_db 方法为当前连接重新选择一个默认数据库。语法格式如下：

```
bool select_db(string $dbname)
```

参数$dbname 指定要设置的默认数据库的名称。如果设置成功，方法返回 true；否则方法返回 false。

设置默认数据库可以为后续访问数据库操作提供方便。比如在编写 SQL 语句时，可以用简单表名表示默认数据库中的一个表，而不需要用数据库名来限制。

3. 执行 SQL 语句

可以调用 MySQLi 对象的 query 方法向服务器发送并执行指定的 SQL 语句。语法格式如下：

```
mixed query(string $sql [, int $resultmode=MYSQLI_STORE_RESULT])
```

参数$sql 指定要执行的 SQL 语句。如果 SQL 语句执行失败，方法返回 false。如果成功执行 SELECT 等查询语句，方法返回一个 MySQLi_RESULT 对象。如果成功执行 INSERT 等数据更新语句，方法返回 true。

参数$resultmode 可以取以下两个常量之一，默认值为 MYSQLI_STORE_RESULT。

（1）MYSQLI_STORE_RESULT。默认值。将查询结果作为一个缓存集返回。该选项会增加内存需求，但可以让用户更方便地处理整个结果集。

（2）MYSQLI_USE_RESULT。将查询结果作为一个非缓存集返回。对于较大的结果集，该选项能提高性能，但对结果集的一些操作会受到限制，如无法立即确定查询结果的行数，无法直接跳到特定的行等。

4. 获取受影响的行数

访问 MySQLi 对象的$affected_rows 属性，可以获取最近一次执行 SQL 语句而受影响的行数。

```
int $affected_rows
```

如果最近一次执行的是 INSERT、UPDATE 和 DELETE 等 SQL 语句，该属性返回插入、更新和删除的行数。

如果最近一次执行的是 SQL SELECT 语句，该属性返回查询结果的行数。此时有一个条件，即在使用 query 方法执行 SELECT 语句时，第 2 个参数必须选择默认值，否则该属性不能返回正确的结果。

【例 8-2】 执行 SQL INSERT 语句，获取受影响的行数。这里假定$mysqli 是一个已经创建的 MySQLi 对象，代表与数据库 elective_manage 的一个连接。代码如下：

```
<?php
header("Content-type:text/html;charset=UTF-8");
… // 创建与数据库的连接
$mysqli->query("SET NAMES 'utf8'");
$sql = "INSERT INTO opencourse VALUES(null, '2015-2016-1', '0901011008',
'1011', default),"
        . "(null, '2015-2016-1', '0901011008', '1078', default)";
$ret = $mysqli->query($sql);
if(!$ret) {
   echo "SQL 语句执行失败！";
   return;
}
echo "插入的行数：", $mysqli->affected_rows;

$mysqli->close();
?>
```

8.1.3 处理查询结果

通过调用 MySQLi 对象的 query 方法，可以执行一条 SQL SELECT 等查询语句，并获得一个 MySQLi_RESULT 对象，又称结果集对象。利用 MySQLi_RESULT 类定义的属性和方法，可以访问并处理结果集。

1. 获取列数和行数

访问结果集对象的$field_count 属性，可以获取结果集包含的列数。

```
int $field_count
```

访问结果集对象的$num_rows 属性，可以获取结果集包含的行数。

```
int $num_rows
```

此属性仅适用于缓存结果集。

2. 移动游标

每个结果集都有一个游标指向某一行，游标所指的行称为当前行。当结果集对象刚返回时，游标指向第一行（偏移量为 0）。可以调用结果集对象的 data_seek 方法移动游标。语法格式如下：

```
bool data_seek(int $offset)
```

参数$offset 的有效取值范围为 0~$num_rows–1。

当参数$offset 的值有效时，方法移动游标至指定的行并返回 true；否则，方法返回 false。该方法仅适用于缓存结果集。

3. 以对象形式返回结果集的一行

调用结果集对象的 fetch_object()方法可以返回一个包含当前行数据的对象。语法格式如下：

```
object fetch_object()
```

方法获取结果集中当前行（游标所指行）的各列数据，并把数据保存在一个对象中返回，游标移至下一行。如果没有更多的行，方法返回 NULL。

当前行的每个数据在对象中被设置为一个属性，属性名为该数据所在列的列名（区分大小写），属性值则为该数据的字符串形式。数据库中的 NULL 表示为 PHP 的 NULL 值。

4. 以数组形式返回结果集的一行

调用结果集对象的 fetch_array()方法可以返回一个包含当前行数据的数组。语法格式如下：

```
mixed fetch_array([int $resulttype = MYSQLI_BOTH])
```

方法获取结果集中当前行（游标所指行）的各列数据，并把数据保存在一个数组中返回，游标移至下一行。如果没有更多的行，方法返回 NULL。

当前行的每个数据在数组中保存为一个或两个元素，元素值总是该数据的字符串形式，元素键可以是列名，也可以是列号，或者两者都存在。这取决于参数$resulttype 的设置。

参数$resulttype 可以取以下常量，其中默认值为 MYSQLI_BOTH：

（1）MYSQLI_ASSOC：返回一个关联数组，元素键为列名。

（2）MYSQLI_NUM：返回一个数字索引数组，元素键为列号。

（3）MYSQLI_BOTH：返回一个数组，既是关联数组又是数字索引数组。

默认情况下，方法返回的数组既是关联数组又是数字索引数组，所以既可以通过列名又可通过列号访问指定列的数据。列号的取值范围是 0~$field_count–1。

5. 释放查询结果

一旦完成对结果集的处理，应该调用结果集对象的 free()方法，释放结果集所占用的内存空间。语法格式如下：

```
void free(void)
```

注意：调用该方法后，结果集对象就不再可用了。

【例 8-3】 执行 SQL SELECT 语句，然后访问结果集。这里假定$mysqli 是一个已经创建的数据库连接对象。代码如下：

```
<?php
...      // 创建与数据库的连接
$mysqli->query("SET NAMES 'utf8'");
$query = "SELECT sn, sname, birthday FROM student WHERE gender='男'";
$result = $mysqli->query($query);
if(!$result) {
   echo "SQL 语句执行失败！";
   return;
}
echo "共" . $result->num_rows . "行：<br />";
while($row = $result->fetch_array()) {
   echo $row['sn'] . " " . $row['sname'] . " ". $row['birthday']."<br />";
```

```
13. }
14.
15. $result->free();
16. $mysqli->close();
17. ?>
```

MySQLi_RESULT 类实现了 Traversable 接口，所以 MySQLi_RESULT 对象是一种可遍历对象。可以利用 foreach 语句来访问其中的每一行。该例代码中的第 11~13 行也可以用下面代码替换：

```
foreach($result as $row) {
   echo $row['sn'] . " " . $row['sname'] . " " . $row['birthday'] . "<br />";
}
```

8.1.4 事务管理

这里介绍 MySQLi 对象的有关事务管理的相关方法。

1. 设置自动提交模式

默认情况下，连接处于自动提交模式，即一旦执行一条更新数据库的 SQL 语句，更新马上反映到数据库。这种改变是永久的，不可回滚。

调用 MySQLi 对象的 autocommit 方法，可以改变连接的自动提交模式。语法格式如下：

```
bool autocommit(bool $mode)
```

如果参数$mode 指定为 true，则设置连接为自动提交模式；若指定为 false，则设置连接为非自动提交模式。

当将自动提交模式改为非自动提交模式时，之后所做的所有 SQL 操作都被视为属于一个事务，直至调用 commit 方法提交事务，或者调用 rollback 方法回滚事务。然后一个新的事务重新开始。

2. 初始化一个事务

除了 autocommit 方法，也可以调用 MySQLi 对象的 begin_transaction 方法来设置非自动提交模式。语法格式如下：

```
bool begin_transaction(void)
```

无论方法调用前是否为自动提交模式，方法调用后都将改变为非自动提交模式，直至调用 commit 方法提交事务，或者调用 rollback 方法回滚事务。然后恢复 begin_transaction 方法调用前的提交模式（自动或非自动）。

3. 提交事务

MySQLi 对象的 commit 方法用于将当前事务提交给数据库。如果成功提交，方法返回 true；否则方法返回 false。语法格式如下：

```
bool commit(void)
```

4. 回滚事务

MySQLi 对象的 rollback 方法用于回滚当前事务。如果成功回滚，方法返回 true；否则方法返回 false。语法格式如下：

```
bool rollback(void)
```

【例 8-4】 一个事务内包含若干数据更新操作。代码如下：

```
<?php
… // 创建与数据库的连接
$data = array(array('201209031001', 1, 80), array('201209031002', 1, 90));
$mysqli->begin_transaction();
foreach($data as $row) {
    $sql = "UPDATE elective SET score=$row[2] WHERE sn='$row[0]' AND lid=$row[1]";
    $mysqli->query($sql);
}
$mysqli->commit();
echo "数据已成功更新！";
$mysqli->close();
?>
```

这里，$data 是一个二维数组，其中每个内部数组包含学号、开课号和成绩 3 个数据。代码利用已经创建的连接对象$mysqli 对学生选课表 elective 中的有关数据做相应的更新。

8.2 使用预处理语句

预处理语句通常是指包含待定参数、经过事先预处理的 SQL 语句。预处理语句适合需要多次执行，每次执行只需指定待定参数值的情况。

MySQLi_STMT 类的一个实例对象代表一个预处理语句，简称语句对象。利用该对象具有的方法，可以为预处理语句设置参数、执行预处理语句并获取执行结果。

8.2.1 创建预处理语句

可以分两步来创建预处理语句：先是初始化一个语句对象，然后再准备一个要执行的 SQL 语句。

1. 初始化语句对象

在创建和执行 SQL 预处理语句之前，首先需要初始化一个语句对象，即 MySQLi_STMT 类的一个实例。调用 MySQLi 对象的 stmt_init 方法可以初始化一个语句对象。语法格式如下：

```
mysqli_stmt stmt_init()
```

方法产生、初始化一个语句对象并返回。

2. 准备 SQL 语句

调用 MySQLi_STMT 对象的 prepare 方法，准备一个 SQL 语句。语法格式如下：

```
bool prepare(string $sql)   // 准备一个 SQL 语句以便执行
```

参数$sql 是一个字符串，指定待执行的 SQL 语句，方法实现对 SQL 语句的预处理。这里，SQL 语句可以在合适的位置包含一个或多个参数标记（?），在执行预处理语句之前，应该给语句中的参数设置具体的值。

如果成功创建 SQL 预处理语句，方法返回 true；否则方法返回 false。

8.2.2 执行预处理语句

在执行 SQL 预处理语句之前，首先需要通过绑定参数为预处理语句中的参数设置具体的值。

1. 绑定参数

可以调用 MySQLi_STMT 对象的 bind_param 方法为 SQL 预处理语句中的参数标记设置参数值。语法格式如下：

```
bool bind_param(string $types, mixed &$var1 [, mixed &$var2]*)
```

其中，参数$types 是一个字符串，其中的每个字符表示后面对应参数变量的数据类型，目前支持 4 种类型。

- i：整数类型（integer）。
- d：浮点数类型（double）。
- s：字符串类型。
- b：Blob 类型。

传递给$var1、$var2 等参数的应该是 PHP 变量，且按引用传递。方法将$var1、$var2 等变量依次绑定至预处理语句中对应的参数标记，即用变量的值设置相应的参数值。

参数$types 的长度、变量的个数以及预处理语句中参数标记的个数必须一致。

如果成功绑定参数，方法返回 true；否则方法返回 false。

2. 执行

调用 MySQLi_STMT 对象的 execute 方法可以执行已经设置了参数的 SQL 预处理语句。语法格式如下：

```
bool execute()  // 执行准备好的 SQL 语句
```

如果 SQL 预处理语句成功执行，方法返回 true；否则方法返回 false。

3. 产生缓存结果集

每次执行完 SQL SELECT 预处理语句，可以调用 MySQLi_STMT 对象的 store_result 方法获得一个缓存结果集。语法格式如下：

```
bool store_result(void)
```

成功执行时，方法返回 true；否则方法返回 false。

4. 获取受影响的行数

访问 MySQLi_STMT 对象的$affected_rows 属性，可以获取最近一次执行 SQL 预处理语句而受影响的行数。语法格式如下：

```
int $affected_rows
```

如果执行的 SQL 预处理语句是 INSERT、UPDATE 或 DELETE 等语句且执行成功，属性返回插入、更新或删除的行数。

如果执行的 SQL 预处理语句是 SELECT 语句，属性返回查询结果的行数。此时查询结果必须是缓存结果集，否则该属性不能返回正确的结果。

【例 8-5】 用预处理语句完成例 8-4 中的数据更新任务。代码如下：

```
<?php
…                // 创建与数据库的连接
$data = array(array('201209031001', 1, 85.5), array('201209031002', 1, 90.2));
$stmt = $mysqli->stmt_init();
$sql = "UPDATE elective SET score=? WHERE sn=? AND lid=?";
$stmt->prepare($sql);
$stmt->bind_param("dsi", $score, $sn, $lid);
$mysqli->begin_transaction();
foreach($data as $row) {
   $score=$row[2]; $sn=$row[0]; $lid=$row[1];
   $stmt->execute();
}
$mysqli->commit();
echo "数据已成功更新！";
$stmt->close();
$mysqli->close();
?>
```

8.2.3 处理查询结果

在执行了预处理的 SQL SELECT 语句后，可以继续访问和调用 MySQLi_STMT 对象的相关属性和方法对查询结果进行所需的处理。

1. 获取列数和行数

访问 MySQLi_STMT 对象的$field_count 属性，可以获取此次执行的查询结果包含的列数。

```
int $field_count
```

访问 MySQLi_STMT 对象的$num_rows 属性，可以获取此次执行的查询结果包含的行数。

```
int $num_rows
```

此属性仅适用于缓存结果集。

2. 移动游标

执行完预处理的 SQL SELECT 语句后，查询结果的游标一开始指向第 1 行（偏移量为 0）。可以调用 MySQLi_STMT 对象的 data_seek 方法移动游标，其语法格式如下：

```
bool data_seek(int $offset)
```

其中，参数$offset 的有效取值范围为 0~$num_rows–1。

当参数$offset 的值有效时，方法移动游标至指定的行并返回 true；否则，方法返回 false。该方法仅适用于缓存结果集。

3. 绑定结果

在获取查询结果数据之前，首先需要调用 MySQLi_STMT 对象的 bind_result 方法将查询结果的列绑定至相应的 PHP 变量。语法格式如下：

```
bool bind_result(mixed &$var1 [, mixed &$var2]*)
```

传递给$var1、$var2 等参数的应该是 PHP 变量，且按引用传递。方法将查询结果的各列依次绑定至各参数变量。参数变量的数目应该与查询结果的列数相同。如果绑定成功，方法返回 true；否则方法返回 false。

绑定结果操作（bind_result）通常在执行 SQL 预处理语句（execute）之后、获取查询结果（fetch）之前执行。

4. 获取查询结果

调用 MySQLi_STMT 对象的 fetch 方法获取查询结果，其语法格式如下：

```
bool fetch()
```

该方法用于获取游标所指的行并将其各列的值存入绑定的 PHP 变量，游标移至下一行。如果没有更多的行，方法返回 NULL。

5. 释放缓存结果集

调用 MySQLi_STMT 对象的 free_result 方法，可以释放缓存结果集占用的内存空间。语法格式如下：

```
void free_result(void)
```

该方法通常仅在存在缓存结果集的情况下调用。方法调用后，缓存结果集被清空，不可再访问。

6. 关闭语句对象

调用 MySQLi_STMT 对象的 close 方法，可以关闭语句对象。语法格式如下：

```
bool close()
```

close 方法成功执行后，MySQLi_STMT 对象不再可用。

【例 8-6】 利用预处理语句执行 SQL SELECT 语句，并访问查询结果。代码如下：

```
<?php
```

```
… // 创建与数据库的连接
$mysqli->query("SET NAMES 'utf8'");
$stmt = $mysqli->stmt_init();
$sql = "SELECT sn,lid,score FROM elective WHERE score >= ?";
$stmt->prepare($sql);
$var1 = 80.5;
$stmt->bind_param("d", $var1);
$stmt->execute();
$stmt->bind_result($sn, $lid, $score);
while($row = $stmt->fetch()) {
    echo $sn." ".$lid." ".$score."<br />";
}
$stmt->close();
$mysqli->close();
?>
```

8.3　使用 PDO 访问数据库

PDO（PHP Data Object）是一种 PHP 扩展，它的使用方法与 MySQLi 扩展的使用非常相似，但 PDO 不仅能用于访问 MySQL 数据库，也能用于访问其他数据库。

8.3.1　PDO 简介

PDO 是一种数据访问抽象层，它为 PHP 程序员提供了一个轻量级的一致的编程接口。利用它，PHP 应用能够用相同的代码访问各种不同的数据库。这里，PDO 本身并不能直接访问数据库，而是需要被称为 PDO 驱动程序的软件的支持，对数据库的实际访问任务是由具体数据库的 PDO 驱动程序完成的。比如要通过 PDO 访问 MySQL 数据库，就需要一个针对 MySQL 数据库的 PDO 驱动程序作为 PHP 应用（PDO）与 MySQL 之间的桥梁。图 8-1 说明了 PHP 应用、PDO、PDO 驱动程和具体数据库之间关系。

PDO 编程接口由 3 个类组成，包括 PDO 类、PDOStatement 类和 PDOException 类。

8.3.2　建立与数据库服务器的连接

一个 PDO 类的实例对象代表 PHP 应用与数据库服务器之间的一个连接。PDO 类的构造方法的语法格式如下：

```
__construct(string $dsn [, string $username [, string $password]])
```

其中，参数$dsn 称为数据源名称（Data Source Name），包含了与数据库连接所需的各种信息，如服务器的 IP 地址、端口号、数据库的名称等。$dsn 由 PDO 驱动器名开头，紧跟一个冒号，后面的语法通常与具体的 PDO 驱动程序有关。比如，下面表示一个 MySQL 数据库的数据源名称：

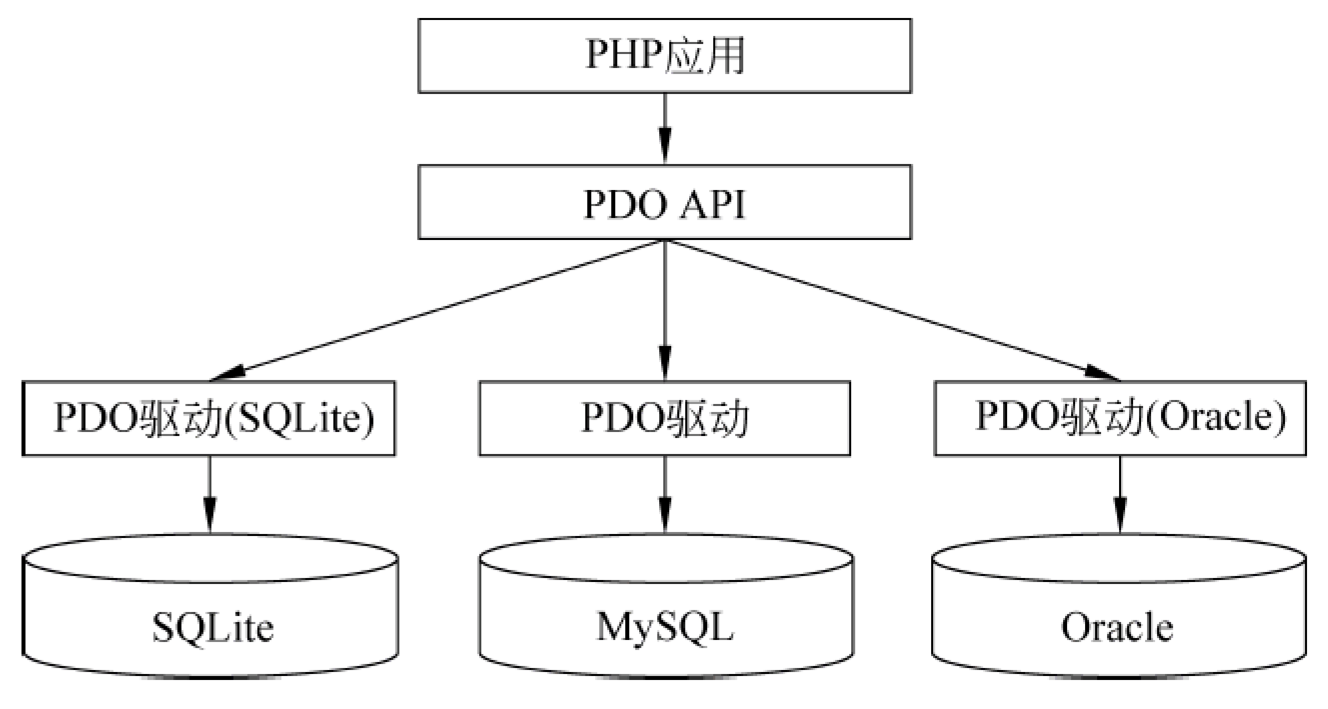


图 8-1 通过 PDO 访问数据库

```
mysql:host=127.0.0.1;port=3306;dbname=elective_manage
```

其中，参数$username 指定用户名，参数$password 指定密码。对于有些 PDO 驱动程序，用户名和密码可能是可选的。

如果连接失败，构造方法将抛出一个 PDOException 型例外。可以利用 try…catch 语句对例外进行捕捉。

【例 8-7】 使用 PDO 连接数据库。代码如下：

```
<?php
header("Content-type:text/html;charset=UTF-8");

$dsn = 'mysql:dbname=elective_manage;host=127.0.0.1';
$user = 'root';
$password = '';
try {
   $pdo = new PDO($dsn, $user, $password);
} catch (PDOException $e) {
   echo '连接失败: ' . $e->getMessage();
   return;
}
echo '成功连接至 elective_manage 数据库';
$pdo = null;
?>
```

8.3.3 执行 SQL 语句

PDO 对象的 exec 和 query 方法可以执行 SQL 语句。

1. 执行增删改

要执行 INSERT、DELETE 和 UPDATE 等 SQL 语句，可以调用 PDO 对象的 exec 方法。

```
int exec(string $statement)
```

如果 SQL 语句执行成功，方法返回受影响的行数。如果没有数据行受影响，方法返回 0。如果 SQL 语句执行失败，方法一般会返回 false。

2. 获取错误码和错误信息

SQL 标准定义了一组执行 SQL 语句产生的状态码，称为 SQLSTATE。SQL 状态码由 5 个字符组成，状态码"00000"表示 SQL 语句被成功执行。

调用 PDO 对象的 errorCode 方法可以获得最近一次通过 PDO 对象执行 SQL 语句产生的状态码。如果没有执行 SQL 语句，方法返回 NULL。

```
mixed errorCode(void)
```

调用 PDO 对象的 errorInfo 方法可以获取更加详细的状态信息。

```
array errorInfo(void)
```

该方法返回一个包含 3 个元素数组，各元素的索引值分别为 0、1 和 2。

- 0：存储 SQLSTATE 码。
- 1：存储特定于 PDO 驱动程序的错误码。
- 2：存储特定于 PDO 驱动程序的错误信息。

【例 8-8】 执行 SQL INSERT 语句并显示受影响的行数。这里假定$pdo 是一个已经创建的 PDO 对象，代表与数据库 elective_manage 的一个连接。

```
<?php
…    // 创建与数据库的连接
$pdo->exec("SET NAMES 'utf8'");
$sql = "INSERT INTO teacher values('1015', '333333','李怀中','会计学院',
DEFAULT)";
$num = $pdo->exec($sql);
if($pdo->errorCode()!=='00000') {
   echo "SQL 语句执行失败！";
   print_r($pdo->errorInfo());
   return;
}
echo "受影响的行数：{$num}。";
?>
```

3. 执行查询

要执行 SQL SELECT 语句，可以调用 PDO 对象的 query 方法。语法格式如下：

```
PDOStatement query(string $statement)
```

如果 SQL 语句被成功执行，方法返回一个 PDOStatement 对象；否则方法返回 false。

在 PDO 扩展中，PDOStatement 对象既表示一个预处理语句，也表示一个查询结果集。PDOStatement 类实现了 Traversable 接口，所以 PDOStatement 对象是一种可遍历对象，可以利用 foreach 语句获取查询结果集的每一行数据。

【例 8-9】 使用 PDO 执行 SELECT 语句并显示查询结果。代码如下：

```
<?php
…   // 创建与数据库的连接
```

```
$pdo->exec("SET NAMES 'utf8'");
$sql = "SELECT tn, tname, dept FROM teacher where tn='2012'";
$pstmt = $pdo->query($sql);      // 执行 SQL SELECT 语句
if($pdo->errorCode()!=='00000') {
   // 执行失败，输出错误信息并返回
   print_r($pdo->errorInfo());
   return;
}
// 遍历结果集
foreach($pstmt as $row) {
   echo $row['tn'], ", ", $row['tname'], ", ", $row['dept'], "<br />";
}
?>
```

8.3.4 使用预处理语句

在 PDO 扩展中，PDOStatement 对象表示一个预处理语句。

1. 创建预处理语句

调用 PDO 对象的 prepare 方法可以创建一个 PDOStatement 对象。语法格式如下：

```
PDOStatement prepare(string $statement)
```

其中，参数$statement 是一个字符串，指定要预处理的 SQL 语句。SQL 语句中往往会包含若干参数。PDO 支持两种类型的参数：命名参数和位置参数。

- 命名参数：以冒号（:）开头，后跟一个字符串，如:name，其中字符串作为参数名称是大小写敏感的。
- 问号参数：一个问号（?）代表一个参数。每个参数以基于 1 的索引编号来标识。

2. 绑定参数

在执行预处理语句之前，要先为语句中的参数指定值。指定参数有两种方式：绑定变量至参数和绑定值至参数。

绑定变量至参数可以调用 PDOStatement 对象的 bindParam 方法，其语法格式如下：

```
bool bindParam(mixed $parameter, mixed &$variable [, int $type])
```

其中，参数$parameter 指定 SQL 预处理语句中的参数。对命名参数，使用参数的名称，如:name；对问号参数，使用参数的编号，例如 1。参数$variable 指定 PHP 变量。方法将用 PHP 变量的值作为参数值，当多次执行预处理语句时，只须改变变量的值即可，不需要再次绑定。

参数$type 显式指定参数的数据类型，可以是以下常量。

- PDO::PARAM_BOOL：SQL BOOLEAN 类型。
- PDO::PARAM_INT：SQL INTEGER 类型。
- PDO::PARAM_NULL：SQL NULL 类型。
- PDO::PARAM_STR：SQL 字符串类型。

绑定值至参数可以调用 PDOStatement 对象的 bindValue 方法，其语法格式如下：

```
bool bindValue(mixed $parameter, mixed $value [, int $type])
```

其中，参数$parameter 指定 SQL 预处理语句中的参数，参数$value 指定要为参数设置的值，参数$type 指定参数的数据类型。

与绑定变量不同，在采用绑定值的方式下，当多次执行预处理语句时，每次都应该调用该方法为参数指定值。

3. 执行预处理语句

调用 PDOStatement 对象的 execute 方法可以执行预处理语句。语法格式如下：

```
bool execute()
```

如果预处理语句包含参数，应该事先调用 bindParam 或 bindValue 方法设置参数值。

如果预处理语句被成功执行，方法返回 true；否则方法返回 false。也可以在执行预处理语句后，通过调用 PDOStatement 对象的 errorCode 和 errorInfo 方法获取相应的错误码和错误信息。

errorCode 方法用于返回最近一次执行预处理语句产生的状态码，该状态码的含义与 PDO 对象的 errorCode 方法返回的状态码的相同。

```
string errorCode(void)
```

errorInfo 方法用于返回最近一次执行预处理语句产生的状态信息，该状态信息的组成和含义与 PDO 对象的 errorInfo 方法返回的状态信息的相同。

```
array errorInfo(void)
```

如果最近执行的预处理语句是 INSERT、DELETE 或 UPDATE 等 SQL 语句，那么可以调用 PDOStatement 对象的 rowCount 方法获取受影响的行数。

```
int rowCount(void)
```

【例 8-10】 使用 PDO 创建并执行预处理语句。代码如下：

```
<?php
… // 创建与数据库的连接
$sql = "SELECT sn, sname, birthday FROM student where gender=?";
$pstmt = $pdo->prepare($sql);  // 创建预处理语句
$genders = array("女", "男");
$pstmt->bindParam(1, $gender); // 绑定变量至参数
foreach($genders as $gender) {
   if(!$pstmt->execute()) {     // 执行预处理语句
      // 执行失败，输出错误信息并返回
      print_r($pstmt->errorInfo());
      return;
   }
   // 遍历结果集
   foreach($pstmt as $row) {
```

```
        echo $row['sn'], ",", $row['sname'], ",", $row['birthday'], "<br />";
    }
}
?>
```

8.3.5 访问查询结果集

在 PDO 扩展中，PDOStatement 对象既代表一个预处理语句，也代表一个查询结果集。

无论是调用 PDO 对象的 query 方法执行一个 SELECT 语句，还是调用 PDOStatement 对象的 execute 方法执行一个预处理的 SELECT 语句，最终都可以通过使用 foreach 语句遍历 PDOStatement 对象来访问查询结果集。另外，也可以通过调用 PDOStatement 对象的相关方法来访问结果集。

1. 获取列数

调用 PDOStatement 对象的 columnCount 方法可以获取结果集的列数。语法格式如下：

```
int columnCount(void)
```

2. 绑定列

调用 PDOStatement 对象的 bindColumn 方法可以将 PHP 变量绑定至结果集的列。这样，当调用 fetch 方法访问结果集时，可以将列的值自动保存到变量中。语法格式如下：

```
bool bindColumn(mixed $column, mixed &$param [, int $type])
```

参数$column 指定列，可以使用结果集中的列名（区分大小写），也可以使用以 1 开始的列号。参数$param 指定 PHP 变量。

3. 获取结果集的一行

可以调用 PDOStatement 对象的 fetch 方法获取查询结果集的一行。语法格式如下：

```
mixed fetch([int $fetch_style])
```

方法获取游标所指的行，游标移至下一行。如果没有更多的行，方法返回 false。

参数$fetch_style 指定获取模式，决定了方法如何将一行数据返回给调用者。该参数可以为以下常量。

- PDO::FETCH_ASSOC：返回一个关联数组，元素键为列名。
- PDO::FETCH_NUM：返回一个数字索引数组，元素键为以 0 开始的列号。
- PDO::FETCH_BOTH：默认值。返回一个数组，既是关联数组又是数字索引数组。
- PDO::FETCH_BOUND：把各列的值分别赋给事先通过 bindColumn 方法绑定的变量，方法返回 true。

4. 获取一行中的单列值

可以调用 PDOStatement 对象的 fetchColumn 方法获取查询结果集一行中的单列值。语法格式如下：

```
mixed fetchColumn([int $column_number = 0])
```

方法获取游标所指行中指定列的值，游标移至下一行。如果没有更多的行，方法返回false。

参数$column_number 使用以 0 开始的列号，指定要获取值的列，默认值为 0，即获取第 1 列的值。

5. 获取所有行

调用 PDOStatement 对象的 fetchAll 方法可以获取查询结果集的所有行，并返回一个数组。语法格式如下：

```
array fetchAll([int $fetch_style [, mixed $fetch_argument]])
```

参数$fetch_style 指定获取模式，可以取以下常量。

- PDO::FETCH_ASSOC：内层数组的元素键为列名。
- PDO::FETCH_NUM：内层数组的元素键为以 0 开始的列号。
- PDO::FETCH_BOTH：默认值。内层数组的元素键为列名和以 0 开始的列号。
- PDO::FETCH_COLUMN：返回指定列的所有值。

如果$fetch_style 指定为 PDO::FETCH_COLUMN，那么参数$fetch_argument 应该使用以 0 开始的列号，指定要获取值的列。在这种情况下，方法返回一个一维数组，元素的键是以 0 开始的整数键，元素的值就是结果集在指定列上的各值。

在其他情况下，方法返回一个二维数组，外层数组都是以 0 开始的整数键，内层数组的元素键则取决于具体的获取模式。

【例 8-11】 使用 PDO 执行预处理语句，并访问查询结果集。代码如下：

```
<?php
… // 创建与数据库的连接
$pdo->exec("SET NAMES 'utf8'");
$sql = "SELECT term, cname, score FROM elective, opencourse, course "
        . "WHERE elective.lid=opencourse.lid AND opencourse.cn=course.cn"
        . "AND sn=:sn ORDER BY 1";
$pstmt = $pdo->prepare($sql);   // 创建预处理语句
$sn = "201209031004";
$pstmt->bindValue(":sn", $sn); // 绑定值至参数
$pstmt->execute();
$result = $pstmt->fetchAll(PDO::FETCH_NUM);
foreach($result as $row) {      // 遍历一个二维数组
   echo $row[0], ", ", $row[1], ", ", $row[2], "<br />";
}
$pstmt = null;
$pdo = null;
?>
```

8.3.6 管理事务

在 PDO 扩展中，事务管理由 PDO 对象的相关方法来完成。默认情况下，PDO 连接处于自动提交模式。

1. 初始化一个事务

PDO 对象的 beginTransaction 方法用于关闭连接的自动提交模式，开始一个事务。语法格式如下：

```
bool beginTransaction(void)
```

事务将持续至调用 commit 方法或者调用 rollback 方法结束，然后恢复 PDO 连接的自动提交模式。

2. 提交事务

PDO 对象的 commit 方法用于将当前事务提交给数据库。成功时方法返回 true；否则方法返回 false。语法格式如下：

```
bool commit(void)
```

3. 回滚事务

PDO 对象的 rollback 方法用于回滚当前事务。成功时方法返回 true；否则方法返回 false。语法格式如下：

```
bool rollBack(void)
```

【例 8-12】 用 PDO 事务和预处理语句完成例 8-4 和例 8-5 中的数据更新任务。代码如下：

```
<?php
… // 创建与数据库的连接
$data = array(array('201209031001', 1, 80), array('201209031002', 1, 90));
$sql = "UPDATE elective SET score=:score WHERE sn=:sn AND lid=:lid";
$pstmt = $pdo->prepare($sql);
$pstmt->bindParam(":score", $score);
$pstmt->bindParam(":sn", $sn);
$pstmt->bindParam(":lid", $lid);
$pdo->beginTransaction();
foreach($data as $row) {
    $score=$row[2]; $sn=$row[0]; $lid=$row[1];
    $pstmt->execute();
}
$pdo->commit();
echo "数据已成功更新！";
$pstmp = null;
$pdo = null;
?>
```

8.4 分 页 显 示

在 Web 应用中，当要显示大型数据集时，通常会利用表格元素<table>进行分页显示。也就是说，每次传送数据集的一部分数据（即一页数据）至客户端，由<table>元素进行呈

现，并在数据底部提供一个翻页导航栏。翻页导航栏通常由一些页码超链接组成，当用户单击这些页码超链接时，将转而呈现上一页、下一页或指定页的数据。这种浏览方式直观、易用，已广泛应用于各种 Web 应用。

为实现分页显示功能，通常需要引入以下变量：

- $rows：数据集的大小，即数据集包含的记录（行）数。
- $pageSize：页面大小，即一次显示几条记录。可由开发人员设置。
- $pageCount：总页数，可以根据$rows 与$pageSize 计算获得，格式如下：

```
$pageCount = (int)ceil($rows/$pageSize);
```

- $currentPage：当前页码。该变量的初值可以设置为 1，之后能由用户指定。
- $first：当前页第 1 条记录在数据集中的索引。可以根据$currentPage 和$pageSize 计算获得，格式如下：

```
$first = ($currentPage-1)*$pageSize;
```

要显示的数据集通常通过查询数据库获得。在实现分页显示时，可以有两种策略：一种是每次只从数据库获取当前页的数据并呈现；另一种是先从数据库获取整个数据集，然后再呈现当前页的数据。

下面介绍在 PHP 中，实现两种策略的相关代码。在第一种策略中，可以先使用类似下面的代码，获取数据集的大小：

```
/* 查询语句，获取满足条件的记录数 */
$query1 = "SELECT COUNT(*) FROM … WHERE … ";
/* 调用数据库连接对象$mysqli 的 query 方法执行查询语句 */
$result1 = $mysqli->query($query1);
/* 获取数据集大小 */
$rows = $result1->fetch_array()[0];
```

在每次显示页面数据时，可以先计算当前页第 1 条记录在数据集中的索引$first，然后利用 SELECT 语句的 LIMIT 短语获取当前页的数据。格式如下：

```
$query2 = "SELECT * FROM … WHERE … ORDER BY … LIMIT $first, $pageSize";
$result2 = $mysqli->query($query2);
```

在这种策略的实现中，应该确保查询时各记录的排列顺序是唯一确定的。最后，利用类似下面代码遍历查询结果并将其呈现出来：

```
while($row = $result2->fetch_array()) { … }
```

在第二种策略中，可以使用类似下面的代码，获取整个数据集及其大小：

```
$query = "SELECT * FROM … WHERE … ORDER BY …";
$result = $mysqli->query($query);
$rows = $result->num_rows;
```

在每次显示页面数据时，应该先计算当前页第 1 条记录在数据集中的索引$first，然后

利用查询结果集对象$result 的 data_seek 方法定位游标，最后依次读取$pageSize 条记录并将其呈现出来：

```
$first = ($currentPage-1)*$pageSize;
$result->data_seek($first);
$count = 1;
while(($row = $result->fetch_array()) && $count++<=$pageSize) { … }
```

【例 8-13】 创建如图 8-2 所示的翻页导航栏。除当前页页码（示意图中为 7），其他页码都是超链接文字（显示边框），当前页页码为普通文本（粗体显示）。单击某个页码超链接，将产生对当前页面文件的请求，并以单击的页码为请求参数，单击的页码将成为当前页页码。如果当前页页码为 1，“上一页”超链接不显示。如果当前页页码为最后一个页码（如示意图中的 13），“下一页”超链接不显示。



图 8-2　翻页导航栏示意图

在翻页导航栏中，除了显示第 1 页和最后一页的页码外，还会显示包含当前页页码在内的若干连续页码超链接。这个连续页码超链接的数量可以设置，在示意图中为 5（包括当前页页码）。

该例结果包含 3 个文件：一个是页面文件，文件名为 8-13(pager).php；第二个是函数库文件，文件名为 pager.php；最后一个是外部样式表文件，文件名为 pager.css。

下面是页面文件 8-13(pager).php 的代码。其功能主要是实验性地设置数据集的大小、页面大小以及连续页码超链接数等参数，然后调用函数库中的相关函数产生翻页导航栏的 HTML 代码，并依据外部样式表中的样式将其呈现出来：

```
<link rel='stylesheet' type='text/css' href='pager.css'/>
<?php
  include_once 'pager.php';
  $rows = 100;
  $pageSize = 8;
  $pageCount = (int)ceil($rows/$pageSize);
  @ $currentPage = $_GET['p'];
  if(empty($currentPage)) $currentPage = 1;
  elseif($currentPage<1) $currentPage = 1;
  elseif($currentPage>$pageCount) $currentPage = $pageCount;
  $showPages = 5;      // 连续页码超链接数
  list($startPage, $endPage) = getBounds($pageCount, $currentPage,
  $showPages);
  $url = $_SERVER['SCRIPT_NAME'];
```

```
    $links = getLinks($startPage, $endPage, $pageCount, $currentPage, $url);
?>
<div class="pager">
    <?php echo $links; ?>
</div>
```

库文件 pager.php 包含两个函数。函数 getBounds 用于获得连续页码超链接的最小页码和最大页码。下面函数 getLinks 用于产生翻页导航栏 HTML 代码：

```
<?php
/*
 * 功能：获得最小页码和最大页码
 * 参数：$pageCount：总页数
 *        $currentPage：当前页码
 *        $showPages： 连续的页码超链接数
 * 返回：pages[0]为最小页码，pages[1]为最大页码
 */
function getBounds($pageCount, $currentPage, $showPages) {
    $startPage = $currentPage-(int)(($showPages-1)/2);  //初步的最小页码
    $endPage = $currentPage+(int)($showPages/2);         //初步的最大页码
    $s = 0;   // 最小页码的偏差
    if($startPage<1) {
        $s = -$startPage+1;
        $startPage = 1;
    }
    $e = 0;   // 最大页码的偏差
    if($endPage>$pageCount) {
        $e = $endPage-$pageCount;
        $endPage = $pageCount;
    }
    $startPage = max($startPage-$e, 1);     //用最大页码的偏差去调整最小页码
    $endPage = min($endPage+$s, $pageCount);//用最小页码的偏差去调整最大页码
    $pages = array();
    array_push($pages, $startPage);
    array_push($pages, $endPage);
    return $pages;
}
/*
 * 功能：获得翻页导航栏的 HTML 代码
 * 参数：
 *    $startPage：最小页码
 *    $endPage：最大页码
 *    $pageCount：总页数
 *    $currentPage：当前页码
 *    $url：页码超链接要访问的页面的 URL（可以包含请求参数）
```

```
*/
function getLinks($startPage,$endPage, $pageCount, $currentPage, $url){
  if(strpos($url, "?")) $pre = "&amp;"; else $pre = "?";
  $links = "";
  if($currentPage>1) {
    $dpage = $currentPage-1;
    $links .= "<a href='$url{$pre}p=$dpage'>上一页</a>";
  }
  if($startPage>1) {
    $links .= "<a href='$url{$pre}p=1'>1</a>";
  }
  if($startPage>2) {
    $links .= "<span>…</span>";
  }
  for($i= $startPage; $i<=$endPage; $i++) {
    if($i==$currentPage) {
      $links .= "<span class='current'>$i</span>";
    } else {
      $links .= "<a href='$url{$pre}p=$i'>$i</a>";
    }
  }
  if($endPage<$pageCount-1) {
    $links .= "<span>…</span>";
  }
  if($endPage<$pageCount) {
    $links .= "<a href='$url{$pre}p=$pageCount'>$pageCount</a>";
  }
  if($currentPage<$pageCount) {
    $dpage = $currentPage+1;
    $links .= "<a href='$url{$pre}p=$dpage'>下一页</a>";
  }
  return $links;
}
?>
```

在产生的翻页导航栏 HTML 代码中，当前页的页码用<span>元素呈现，且 class 属性值必须包含类名'current'，以便使用外部样式表中定义的样式。外部样式表文件 pager.css 包括“基本样式”和“翻页导航栏样式”两部分，其中“基本样式”部分已经在例 2-19 中给出，并且作为了实例应用的基本样式，下面是“翻页导航栏样式”部分的代码：

```
/* 翻页导航栏样式 */
.pager a {
  float: left; padding: 2px 3px; margin: 2px 3px; color: steelblue;
  border: 1px outset steelblue; font-size: 13px
}
```

```
.pager span {
   float: left; padding: 2px 3px; margin: 2px 3px; color: #458994
}
.pager span.current {
   margin: 3px 4px; font-weight: 700
}
```

8.5　实例：浏览教师信息

本实例实现管理员子系统导航栏中“浏览教师信息”菜单项的功能，可以分页呈现所有教师的信息，如图 8-3 所示。



图 8-3　浏览教师信息

首先介绍本实例涉及的一些 CSS 样式和函数库。除了之前实例引入的、存放在外部样式表文件 xk.css 中“基本样式”和“水平导航栏样式”，本实例新引入“翻页导航栏样式”和“数据表格样式”。“翻页导航栏样式”在前面例 8-13 中已经给出，这里不再赘述。“数据表格样式”定义了如下 CSS 规则：

```
/* 数据表格样式 */
table {border-width: 0; margin: 10px 0; border-collapse: collapse;
font-size: 14px}
tr {height: 30px}
thead {
   color: #458994; border-top: 1px solid #458994; border-bottom: 2px solid
  #458994
}
tfoot {color: #458994; border-top: 1px solid #458994}
```

与之前引入的样式一样，“翻页导航栏样式”和“数据表格样式”也都保存到外部样式表文件 xk.css 中，该文件处于教务选课系统项目 xk 中的“源文件”结点下的 css 子目录。

本实例涉及两个函数库文件：一个是在例 8-13 引入的 pager.php；还有一个是有关数据库连接和访问的 mysql.php。本实例用到库文件 mysql.php 中 connect 和 executeSql 两个函数，下面是它们的代码：

```
<?php
/*
 * 功能：创建与数据库 elective_manage，返回连接对象，
 *       调用者应访问连接对象的相关属性判别是否连接成功
 * 输入：无
 */
function connect() {
   @$mysqli = new mysqli('localhost', 'root', '', 'elective_manage');
   return $mysqli;
}
/*
 * 功能：执行 SQL 语句，返回执行结果。如果执行失败，返回 false；
 *       如果成功执行 INSERT 等数据更新语句，返回 true；
 *       如果成功执行 SELECT 等查询语句，返回一个 MySQLi_RESULT 对象
 * 输入：$mysql：与数据库的连接对象
 *       $sql：要执行的 SQL 语句
 */
function executeSql($mysqli, $sql) {
   $mysqli->set_charset('UTF8');
   $result = $mysqli->query($sql);
   return $result;
}
?>
```

这里，把两个函数库文件 pager.php 和 mysql.php 都保存到教务选课系统项目 xk 的“源文件”结点下的 lib 子目录中。

接下来介绍实现“浏览教师信息”功能的页面模块文件 teacher_list.php 和相应的页面文件 teacher_p.php。

模块文件 teacher_list.php 实现分页呈现所有教师信息的功能，其呈现效果如图 8-3 所示页面的主区部分。下面是该模块文件的代码：

```
<!--
*  功能：分页呈现所有教师的信息列表
*  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/
       css/xk.css"/>
-->
<!-- <link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/> -->
<?php
```

```
include_once '../lib/mysql.php';
include_once '../lib/pager.php';
// 连接数据库
$mysqli = connect();
if ($mysqli->connect_errno) {
    echo "不能连接到数据库<br/>";
    return;
}
/* 通过查询数据库获取总行数，设置页面大小，并计算总页数 */
$query = "SELECT COUNT(*) FROM teacher";
$result = executeSql($mysqli,$query);
$rows = $result->fetch_array()[0];
$pageSize = 3;  // 设置页面大小
$pageCount = (int)ceil($rows/$pageSize);   // 总页数
/* 确定当前页码 */
@ $currentPage = $_GET['p'];
if(empty($currentPage)) $currentPage = 1;
elseif($currentPage<1) $currentPage = 1;
elseif($currentPage>$pageCount) $currentPage = $pageCount;
/* 构建翻页导航栏的 HTML 代码或者是相当的信息，并保存在变量$links 中 */
$links = "";
if($rows === 0) {
    $links = "<span>无数据！</span>";
} elseif($pageCount === 1) {
    $links = "<span>共{$rows}条记录！</span>";
} else {
    $showPages = 1;  // 设置翻页导航栏中连续页码超链接数
    list($startPage, $endPage) = getBounds($pageCount, $currentPage, $showPages);
    $url = $_SERVER['SCRIPT_NAME'];  // 当前被请求 PHP 文件路径和文件名
    $links = getLinks($startPage, $endPage, $pageCount, $currentPage, $url);
}
/* 获取当前页的数据记录 */
$first = ($currentPage-1)*$pageSize;
$query = "SELECT * FROM teacher ORDER BY dept, tn LIMIT $first, $pageSize";
$result = executeSql($mysqli,$query);
?>
<!-- 呈现当前页教师数据 -->
<style type="text/css">
    table .c1 {width: 80px; padding-left: 10px}
    table .c2 {width: 80px}
    table .c3 {width: 180px}
    table .c4 {width: 100px}
</style>
```

```
<table>
  <thead>
    <tr>
    <td class="c1">职工号</td><td class="c2">姓名</td>
    <td class="c3">所属部门</td><td class="c4">是否管理员</td>
    </tr>
  </thead>
  <tbody>
    <?php while($row = $result->fetch_array()) { ?>
      <tr>
      <td class="c1"><?php echo $row['tn'] ?></td>
      <td class="c2"><?php echo $row['tname'] ?></td>
      <td class="c3"><?php echo $row['dept'] ?></td>
      <td class="c4"><?php echo $row['admin']==='是'?'√':'' ?></td>
      </tr>
    <?php } ?>
  </tbody>
  <tfoot>
    <tr style="height: 10px"><td colspan="4"></td></tr>
  </tfoot>
</table>
<div class="pager"><?php echo $links ?><div style="clear: both"> </div></div>
<?php $mysqli->close() ?>
```

页面文件 teacher_p.php 利用 PHP 包含文件技术，呈现组成页面的页面头、导航栏、页面主区和页面脚等各部分，其中页面主区由模块文件 teacher_list.php 负责呈现。页面文件的呈现效果如图 8-3 所示，下面是该页面文件的代码：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>浏览教师信息</title>
    <link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
  </head>
  <body>
    <?php include("header_admin.php"); ?>

    <?php
      $name = "刘绍军";
      $choice = 1;
      include("navigation_admin.php");
    ?>

```

```
        <div style="width: 90%; min-height: 150px; margin: 0 auto; padding:
        5px;">
          <?php
             include("teacher_list.php");
          ?>
        </div>

        <?php include("footer_admin.php"); ?>
    </body>
</html>
```

由于还没有实现用户登录和会话等功能，所以还无法动态获取当前登录教师的相关信息。为此，在页面文件调用“导航栏”模块文件时，暂时将该模块文件的输入变量$name设置为某位教师的姓名。在后面的实例中将对此进行改善。

习 题 8

1. 根据要求写 PHP 代码

（1）请通过 MySQLi 扩展建立与 MySQL 数据库 mydb 的连接，产生连接对象$mysqli。其中，数据库服务器的 IP 地址为 127.0.0.1，端口号为 3306，用户名为 liming，密码为 123456。

（2）请通过 MySQLi 扩展建立与 MySQL 数据库服务器的连接，产生连接对象$mysqli。其中，数据库服务器的 IP 地址为 127.0.0.1，端口号为 3306，用户名为 liming，密码为 123456。

（3）假设已经通过 MySQLi 扩展建立了与数据库的连接，连接对象是$mysqli。现在请写出代码，将连接的默认数据库设置为 mydb。

（4）假设已经通过 MySQLi 扩展建立了与数据库的连接，连接对象是$mysqli。现在请写出代码，将连接的默认字符集设置为 utf8。

（5）假设已经通过 MySQLi 扩展建立了与数据库的连接，连接对象是$mysqli。现在请写出代码：执行查询语句"SELECT * FROM student"，然后输出查询结果的行数。

（6）假设已经通过 MySQLi 扩展建立了与数据库的连接，连接对象是$mysqli。现在请写代码，要求执行以下更新语句，然后输出更新的行数：

```
UPDATE student SET password='654321' WHERE sn = '200107211000'
```

（7）假设下面代码已经成功执行，其中$mysqli 表示与数据库的连接对象：

```
$result = $mysqli->query("SELECT * FROM student");
```

现在请写代码，要求获取查询结果第 3 行的数据并放入数组$row 中。

（8）下面 PHP 代码执行一条预处理的 SQL UPDATE 语句，并显示语句执行后受影响的行数。其中，$mysqli 是一个已经创建的数据库连接对象。请根据注释完善下面代码：

```
$stmt = $mysqli->stmt_init();
$sql ="update student set password='123456' where sn = ?";
$stmt->prepare($sql);
```

```
$var1='200107211000';
___________________________ // 将变量$var1 绑定至预处理语句中的参数
$stmt->execute();
___________________________ // 显示受影响的行数
```

（9）下面 PHP 代码创建并执行了一条 SQL SELECT 的预处理语句。现在请写代码为它产生一个缓存结果集：

```
$stmt = $mysqli->stmt_init();
$stmt->prepare("select * from teacher");
$stmt->execute();
```

（10）下面 PHP 代码执行一条预处理的 SQL SELECT 语句，并显示查询结果。其中，$mysqli 是一个已经创建的数据库连接对象。请根据注释完善下面代码：

```
$stmt = $mysqli->stmt_init();
$sql = "select sn,password,name,gender from student where gender = ?";
$stmt->prepare($sql);
$var1 = "男";
_________________________________________ // 将变量$var1 绑定至预处理语句中的参数
$stmt->execute();
_________________________________________ // 将适当变量绑定至查询结果各列
while($row = $stmt->fetch()) {
  echo $sn." ".$password." ".$name." ".$gender."<br />";
}
```

2. 简答题

（1）在 PHP 中，如何利用 MySQLi 扩展实现与 MySQL 数据库的连接？

（2）在 MySQLi 扩展中，如何实现数据库事务管理？

（3）简述 MySQLi_STMT 类中 bind_param 方法和 bind_result 方法的作用及用法。

第 9 章　表单与会话

本章主题：

- 提交表单；
- 表单数据的获取与检验；
- 会话管理；
- 页面跳转与重定向；
- 文件上传；
- 文件下载。

表单和会话是开发 Web 应用的两项基本技术。在客户与 Web 应用交互时，离不开客户向服务器提交有关数据，并由 Web 应用对这些数据进行处理、保存和响应，这需要表单及相关技术的支持。在一个客户与 Web 应用交互时，同样需要会话及相关技术的支持，只有这样，Web 应用才能记住该客户前期交互的状态，并把它作为后续交互的基础。

本章首先介绍表单与表单提交、表单数据的获取与检验等方法和技术，然后介绍会话的概念、会话管理的技术、页面跳转的各种形式以及重定向的概念和实现方法。本章最后介绍文件操作的一些函数，以及文件上传和下载的方法和技术。

9.1　表 单 处 理

表单是用户与 Web 应用进行交互的主要方式。表单用于收集并提交用户数据，服务器接收到用户数据后，可由指定程序进行检验、处理并做出响应。

9.1.1　提交表单

当提交表单时，浏览器将发出 HTTP 请求，表单数据会作为请求参数送往服务器。请求参数的名称对应控件元素的 name 属性值，请求参数的值对应控件元素的 value 属性值。通常有两种方法可以提交表单，即 GET 方法或 POST 方法。

1. GET 方法

在默认情况下，表单是以 GET 方法提交的。也可以将<form>标签的 method 属性设置为"GET"（一般大小写均可），明确要求浏览器以 GET 方法提交表单数据。例如，以下代码显式指明以 GET 方法提交表单：

```
<form method="GET" action="process.php">
  …
  <input type="submit" name="ok" value="注册" />
</form>
```

按 GET 方法提交的表单数据会作为请求参数、以名称-值对的形式出现在 HTTP 请求的

URL 中：

```
… process.php?<参数名 1>=<参数值 1>&<参数名 2>=<参数值 2>...
```

由于 URL 的长度会有一些限制，所以 GET 方法适合少量数据的提交。也由于 GET 方法提交的数据会显示在地址栏上，保密性不好，所以 GET 方法不适合提交密码等敏感数据。

以 GET 方法提交的 HTTP 请求可称为 GET 请求。一般来说，GET 请求适用于查询操作，即从服务器获取 Web 应用的某些状态信息，表单数据则作为查询所需的参数。这种操作通常不会改变服务器端 Web 应用的状态，所以客户端浏览器一般会允许用户不加限制地重复这样的请求，例如反复单击“刷新”按钮。

2．POST 方法

要以 POST 方法提交表单，应该将<form>标签的 method 属性设置为"POST"（一般大小写均可）。例如，以下代码指明以 POST 方法提交表单：

```
<form method="POST" action="process.php">
  …
  <input type="submit" name="ok" value="注册" />
</form>
```

按 POST 方法提交的表单数据会作为请求参数、以名称-值对的形式出现在 HTTP 请求的请求体中：

```
<参数名 1>=<参数值 1>&<参数名 2>=<参数值 2>…
```

与 GET 方法相比较，POST 方法的请求数据放置在请求体中、而不是在 URL 中，所以其信息相对较为安全，且传输的数据量没有大小限制，可以非常大。

以 POST 方法提交的 HTTP 请求可称为 POST 请求。一般来说，POST 请求适用于更新操作，如提交一个订单。这种操作通常会改变服务器端 Web 应用的状态，所以当用户点击“刷新”按钮要重复这样一个请求时，客户端浏览器一般会打开一个对话框，要求用户加以确认。

9.1.2 获取表单数据

在 PHP 中，获取表单数据（即请求参数）的方法非常简单，因为所有的请求参数会被事先保存在系统预定义的超全局变量中。用户代码只需访问这些超全局变量就可以获得所需的请求参数。

1. 获取请求方法

有时候，需要了解本次 HTTP 请求的方法，即是 GET 请求还是 POST 请求。访问超全局变量$_SERVER 可以获取该信息。超全局变量$_SERVER 是一个包含服务器及执行环境信息的数组，要获得当前请求的方法，可以访问以下数组元素。

```
$_SERVER["REQUEST_METHOD"]
```

该元素保存着一个字符串，表示本次 HTTP 请求的方法，如"GET"、"POST"等。

2. 获取 GET 数据

如果当前请求是 GET 请求，那么可以访问超全局变量$_GET 来获得指定请求参数的参数值。格式如下：

```
$_GET[<参数名>]
```

$_GET 是一个包含通过 GET 方法传递给当前脚本的请求参数的数组。用参数名作为元素的键，元素值即为对应的参数值。如$_GET["name"]获取的就是参数名为"name"的参数值。

3. 获取 POST 数据

如果当前请求是 POST 请求，那么可以访问超全局变量$_POST 来获得指定请求参数的参数值。格式如下：

```
$_POST[<参数名>]
```

$_POST 是一个包含通过 POST 方法传递给当前脚本的请求参数的数组。用参数名作为元素的键，元素值即为对应的参数值。如$_POST["name"]获取的就是参数名为"name"的参数值。

有些控件可能有多个值，如复选框组（各复选框的 name 属性值相同）、选择列表（设置了 multiple 属性）。此时，当提交表单时，就会出现多个请求参数的名称-值对，它们具有相同的参数名。在 PHP 中，为了有效读取多值控件的值，控件的名称（控件元素的 name 属性值）应该以一对方括号（[ ]）结尾。

无论是 GET 方法还是 POST 方法，在读取多值参数（或多值控件）的值时，指定的参数名都不要包括最后的方括号（[ ]），但读取的值将是一个字符串数组，而不是一个简单的字符串。

另外需要注意，并不是所有的表单控件都会产生相应的请求参数，如单选按钮（组）、复选框（组）以及多选的选择列表等。对这些控件，若事先没有设置预选项，用户也没有选择，那么提交表单时，就不会有相应的请求参数。在服务器端，为了判断是否存在某个请求参数，可以使用以下数组函数：

```
bool array_key_exists(mixed $key, array $array)
```

该函数判断参数数组$array 中是否存在键为$key 的元素，函数返回一个 boolean 值。比如在一个 POST 请求中，可以用下面函数判断是否存在名为"lang"的参数：

```
array_key_exists("lang", $_POST)
```

如果存在名为"lang"的参数，函数返回 true；否则函数返回 false。

这里，反复涉及数组的有关操作。第 10 章将对数组知识作更加系统和详细的介绍。

【例 9-1】 有一个 PHP 文件，当初始请求时呈现如图 9-1 所示的表单。当用户提交表单时显示用户输入的表单数据。代码如下：

该例结果包含两个文件：一个是 PHP 文件，文件名为 9-1.php；一个是 HTML 文件，文件名为 form1.html。



图 9-1　例 9-1 表单示意图

下面是 form1.html 文件的代码，其作用是呈现表单：

```
<link rel="stylesheet" type="text/css" href="formone.css"/>
<form method="POST">
<div class="outer">
   <div class="title">请输入信息</div>
   <div class="inter">
      <p>
         <label for="i1" class="label">用户名</label>
         <input type="text" id="i1" name="user" maxlength="15"
                  style="width: 130px" />
      </p>
      <p>
         <label for="i2" class="label">密码</label>
         <input type="password" id="i2" name="pw" maxlength="15"
                  style="width: 100px" />
      </p>
      <p>
         <span class="label">性别</span>
         <input type="radio" id="i31" name="gender" value="男"/>
         <label for="i31"/>男</label>
         <input type="radio" id="i32" name="gender" value="女"/>
         <label for="i32">女</label>
      </p>
      <p>
         <span class="label">熟悉的语言</span>
         <input type="checkbox" id="i41" name="lang[]" value="C"/>
```

```
            <label for="i41">C</label>
            <input type="checkbox" id="i42" name="lang[]" value="Java"/>
            <label for="i42">Java</label>
            <input type="checkbox" id="i43" name="lang[]" value="PHP"/>
            <label for="i43">PHP</label>
        </p>
        <p>
            <label for="i5" class="label" style="vertical-align: top">特长</label>
            <select id="i5" name="skill[]" multiple="multiple" size="3"
                    style="width: 80px">
                <option value="1">足球</option>
                <option value="2">篮球</option>
                <option value="3">游泳</option>
            </select>
        </p>
        <p>
            <label for="i6" class="label">email 地址</label>
            <input type="text" id="i6" name="email" style="width: 250px" />
        </p>
        <p style="text-align: center; padding-top: 10px">
            <input type="submit" name="submit" value="提交信息"/>
        </p>
    </div>
</div>
</form>
```

表单的呈现用到了外部样式表文件 formone.css，该样式表文件已经在第 2 章的例 2-20 中提出和介绍，这里不再赘述。

下面是 9-1.php 文件的代码，其主要功能是读取并显示表单数据：

```
<?php
if($_SERVER["REQUEST_METHOD"]=="POST" ) {
    echo "用户名: ", $_POST['user'], "<br />";
    echo "密码: ", $_POST['pw'], "<br />";
    if(array_key_exists("gender", $_POST)) {
        echo "性别: ", $_POST['gender'], "<br />";
    } else {
        echo "性别: ", "未选择", "<br />";
    }
    if(array_key_exists("lang", $_POST)) {
        $value = "";
        foreach($_POST["lang"] as $v) {
            $value .= $v . " ";
        }
```

```
        echo "熟悉的语言：", $value, "<br />";
    } else {
        echo "熟悉的语言：", "未选择", "<br />";
    }
    if(array_key_exists("skill", $_POST)) {
        $value = "";
        foreach($_POST["skill"] as $v) {
            $value .= ($v==="1"?"足球":($v==="2"?"篮球":"游泳")) . " ";
        }
        echo "特长：", $value, "<br />";
    } else {
        echo "特长：", "未选择", "<br />";
    }
    echo "email 地址：", $_POST["email"], "<br />";
    echo "提交按钮：", $_POST["submit"], "<br />";
} else {
    include "form1.html";
}
?>
```

当用户初始请求（GET 请求）该 PHP 文件时，代码装入 form1.html 文件，页面呈现出表单。当用户提交表单时，将再次请求（POST 请求）该 PHP 文件，此时页面显示用户之前在表单中输入的数据。

9.1.3 检验表单数据

读入从客户端发来的用户数据后，接下来应该检验其有效性，只有所有的数据都有效可用，才可以进入具体的业务处理。

首先要检验用户是否提供了一些必要的数据。对表单中的有些控件元素，用户必须提供相应的数据，通常称其为必填项，它们是处理具体业务所必需的。

其次要检验一些数据的格式是否满足相应的要求。如密码的长度是否足够，日期数据中的年、月和日是否有效，email 地址的格式是否符合相关标准的约定等。这方面的检验一般可利用正则表达式工具，即先根据某种数据本身的格式特点构建一个正则表达式，然后调用 preg_match 等函数判断用户数据是否与该正则表达式相匹配。

比如，针对日期数据的格式特点，可以构建以下的正则表达式：

```
$pattern = '/^\d{4}[-](0?[1-9]|1[012])[-](0?[1-9]|[12][0-9]|3[01])$/';
```

如果变量$birthday 保存着用户输入的日期数据，那么就可以用下面函数判断该用户数据是否有效：

```
preg_match($pattern, $birthday)
```

如果函数返回 1，表明用户数据与正则表达式是相匹配的；如果函数返回 0，表明用户数据的格式是有问题的。

类似地，可以用下面函数判断用户输入的电子邮件地址$email 是否有效：

```
preg_match('/^[a-zA-Z0-9_\-.]+@[a-zA-Z0-9\-]+(\.[a-zA-Z0-9\-]+)+$/', 
$email)
```

最后要根据业务背景知识和 Web 应用的状态信息等判断用户数据是否有效。例如，在录入学生成绩时，如果提供的成绩是负数（转换成数值型后），显然是无效的。又比如，在学生登录时，如果用户输入的用户名和密码并不是一个有效的账户信息，那么肯定会拒绝其登录。

如果发现有些用户数据无法通过检验，那么就不能进入业务处理阶段，而只能返回表单由用户重新提供数据。此时，应该向用户提供相应的错误信息，以便用户更改错误。错误信息随表单一同返回客户端，一般可显示在对应控件元素的右侧。例如，以下代码：

```
<input type="text" name="user" … /> <span class="errFont"><?php echo 
$userErr;?></span>
```

其中，<input>元素用于接受用户名，<span>元素用于显示错误信息，变量$userErr 保存着相应的错误信息。当表单初始呈现或者用户输入的用户名没有错，那么变量$userErr 应该是空串，用户名文本域右侧不会有任何信息；否则应该显示相应的错误信息。

另外，在返回表单和显示错误信息的同时，还应该回显用户的错误数据，以便用户在原来的基础上进行修改。例如以下代码：

```
<input type="text" name="user" … value="<?php echo $user; ?>" />
```

其中，用户名文本域的 value 属性值是一个 PHP 代码块，输出变量$user 的值，该值会在文本域中显示。变量$user 的初值可以是空串。当用户提交表单时，该变量应该接收用户输入的用户名。

【例 9-2】 检验表单数据。该例在例 9-1 的基础上修改而成，着重说明如何检验表单数据，如何回显表单数据，以及如何显示错误信息。

该例结果包含两个文件：一个是 PHP 文件 9-2.php，主要实现表单数据的检验；另一个也是 PHP 文件，文件名为 form2.php，用于呈现表单。与例 9-1 中的表单相比，该例表单只包含用户名、性别和 email 地址 3 项，但能够回显用户数据和显示错误信息，如图 9-2 所示。



图 9-2　例 9-2 表单示意图

下面是文件 9-2.php 的代码：

```php
<?php
// 定义变量并设置为空值
$user = $gender = $email = "";
$userErr = $genderErr = $emailErr = "";
$flag = true;

if($_SERVER["REQUEST_METHOD"] == "POST") {
    $user = trim($_POST["user"]);
    if (empty($user)) {
        $userErr = "用户名是必填的";
        $flag = false;
    }
    if(array_key_exists("gender", $_POST)) $gender = $_POST["gender"];
    if(empty($gender)) {
        $genderErr = "请选择性别";
        $flag = false;
    }
    $email = trim($_POST["email"]);
    if (empty($email)) {
        $emailErr = "email 地址不能为空";
        $flag = false;
    } else {
        if(!preg_match('/^[a-zA-Z0-9_\-.]+@[a-zA-Z0-9\-]+(\.[a-zA-Z0-9\-]+)+$/', $email)) {
            $emailErr = "email 地址格式错误";
            $flag = false;
        }
    }
}
if($_SERVER["REQUEST_METHOD"]=="POST" && $flag) {
    echo "用户名：", $user, "<br />";
    echo "性别：", $gender, "<br />";
    echo "email 地址：", $email, "<br />";
} else {
    include "form2.php";
}
?>
```

下面是文件 form2.php 的代码。

```html
<link rel="stylesheet" type="text/css" href="formone.css"/>
<form method="POST">
<div class="outer">
    <div class="title">请输入信息</div>
```

```
  <div class="inter">
    <p>
      <label for="i1" class="label">用户名</label>
      <input type="text" id="i1" name="user" maxlength="15"
                style="width: 130px" value="<?php echo $user ?>" />
      <span class="errMsg"><?php echo $userErr;?></span>
    </p>
    <p>
      <span class="label">性别</span>
      <input type="radio" id="i31" name="gender" value="男"
            <?php if($gender==="男") echo 'checked="checked"' ?> />
      <label for="i31"/>男</label>
      <input type="radio" id="i32" name="gender" value="女"
            <?php if($gender==="女") echo 'checked="checked"' ?>/>
      <label for="i32">女</label>
      <span class="errMsg"><?php echo $genderErr;?></span>
    </p>
    <p>
      <label for="i6" class="label">email 地址</label>
      <input type="text" id="i6" name="email" style="width: 250px"
            value="<?php echo $email ?>" />
      <span class="errMsg"><?php echo $emailErr ?></span>
    </p>
    <p style="text-align: center; padding-top: 10px">
      <input type="submit" name="submit" value="提交信息"/>
    </p>
  </div>
</div>
</form>
```

实际上，对于性别这个单选按钮组，可以设置一个初值，如$gender="男"，这样就无须对其进行检验了，因为总会有一个值。但对用户的选择进行回显还是需要的。

9.2 会 话 管 理

会话是 Web 应用非常重要的概念。这里介绍会话功能的实现原理，以及如何启动会话、维护会话数据等操作。

9.2.1 会话与 Cookie

Web 应用建立在以 HTTP 协议为基础的浏览器-服务器架构之上。HTTP 是一种无状态、无连接的协议，本身并没有会话的概念。Web 服务器不会维持对某个请求的处理结果，也不会将同一个客户的前后几次的请求联系在一起。

然而，Web 应用在客观上需要有会话的功能。比如一个电子商务网站，用户通常可以

前后多次挑选要购买的商品。这就要求服务器能将该用户前后多次的访问联系在一起，能对该用户每一次挑选的商品数据加以保存和维护。

会话是指一个用户对一个网站（或 Web 应用）的一系列请求。这一系列的请求并不要求是连续的，期间可以访问其他的网站（或 Web 应用）。服务器负责维护在会话期间产生的数据，这些数据能够在这一系列的请求处理中共享。

那么如何实现会话功能呢？一般来说，目前的会话功能都是由 PHP 等动态网页技术利用 Cookie 机制实现的。Cookies 是由服务器产生、送往客户端保存的一个个文本数据，每一个 Cookie 数据包括名称、值、失效时间、所属网站（或 Web 应用）等属性。当用户访问某个网站（或 Web 应用）时，相关的 Cookie 数据就会自动随请求一起发往服务器。

为实现会话，PHP 解释器会在会话伊始产生一个所谓的会话 ID，并创建一个包含该数据的 Cookie。在 PHP 中，这个数据的名称默认为 PHPSESSID，也称为会话名。这个 Cookie 数据会随响应一同送往客户端并在客户端保存。之后，当用户再次向该服务器或 Web 应用发送请求时，包含会话 ID 的 Cookie 就会随同请求一起送往服务器。服务器接收到请求后，可以利用其中的会话 ID 判断此次请求处理属于哪个会话，如图 9-3 所示。默认情况下，这种 Cookie 会在浏览器关闭时失效，即被清除。

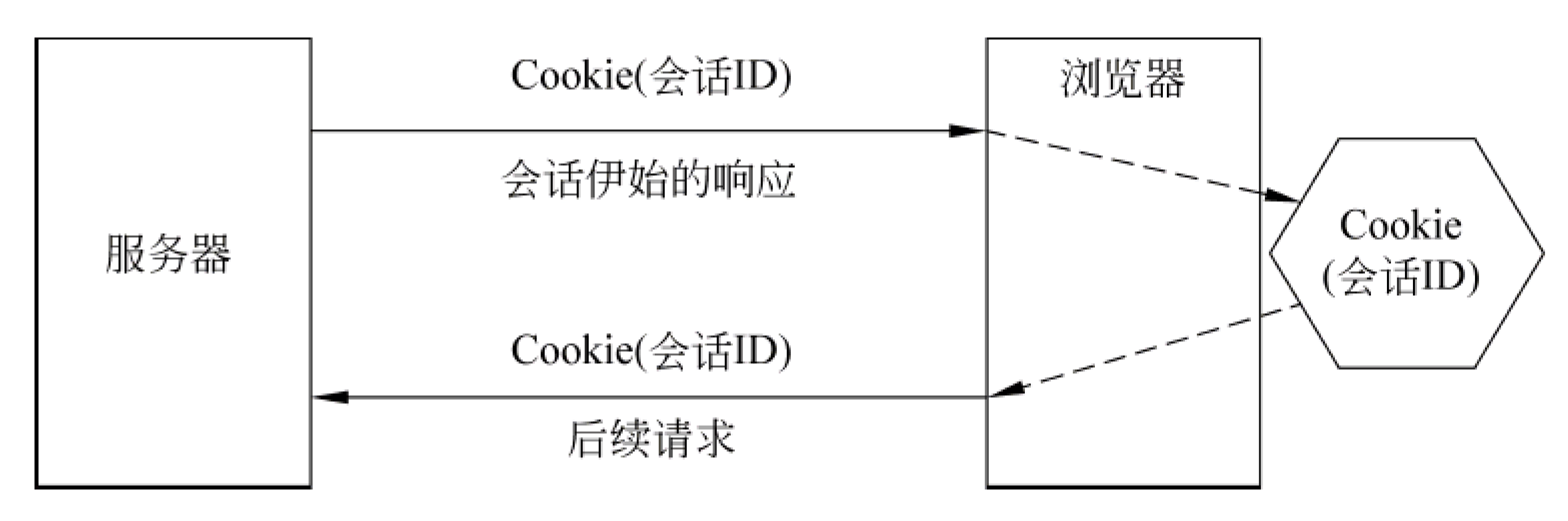


图 9-3　Cookie 与会话

显然，要实现这样的会话功能，客户端浏览器必须支持 Cookie 机制。目前大多数浏览器都是支持的，并且可以由用户进行设置。另外，在 PHP 中，应该将配置文件 php.ini 中的 session.use_cookies 项设置为 1，即

```
session.use_cookies = 1
```

该设置是默认的，它表明要使用 Cookie 机制实现会话：

（1）在会话伊始，PHP 系统会产生会话 ID，并创建一个包含该会话 ID 的 Cookie 送往客户端保存；

（2）在会话过程中，PHP 系统会根据客户端传送过来的会话 ID 匹配相应的会话。

那么怎样开始一个会话呢？通常，可以调用 session_start 函数来启动一个会话：

```
bool session_start()
```

该函数的功能是启动或恢复一个会话。

（1）如果当前请求没有携带包含会话 ID 的 Cookie，那么就启动一个会话：

① 产生一个会话 ID；

② 创建数组$_SESSION，之后的处理中，可以把有关的会话数据保存在该数组里；

③ 响应时，创建一个包含会话 ID 的 Cookie 送往客户端，然后将数组$_SESSION 中的数据保存并使其与会话 ID 建立关联。

（2）如果当前请求携带包含会话 ID 的 Cookie，那么就恢复会话：创建数组$_SESSION，然后恢复与当前会话 ID 相关联的会话数据。

在会话活动状态下，再次调用该函数将被忽略，但会产生一个 Notice 错误信息。

另一种能够启动或恢复会话的变通方法是将配置文件 php.ini 中的 session.auto_start 项设置为 1，即

```
session.auto_start = 1
```

如果这样设置了，就没有必要调用 session_start 函数了，它就如同在每一个 PHP 文件或 PHP 页面中自动包含对函数 session_start()的调用。每当用户请求一个 PHP 文件或 PHP 页面时，PHP 系统就会根据请求中是否包含会话 ID，决定是启动还是恢复一个会话。这个设置不是默认的，默认值为 0。

9.2.2 重写 URL

实现会话功能的另一个途径是重写 URL。所谓重写 URL 就是将会话 ID 附着在页面中超链接<a>等元素的 URL 上。这样，当页面传送到客户端时，会话 ID 也会随着送往客户端。当用户单击超链接等请求指定的 URL 资源时，会话 ID 也就自然回传到服务器。

在 PHP 中，可以将会话 ID 作为一个请求参数附着在 URL 上。这里经常会用到预定义常量 SID。该常量的类型为字符串，格式如下：

```
<session_name>=<session_id>
```

其中，<session_id>即会话 ID，<session_name>是会话 ID 的名称，也称为会话名，其默认值为"PHPSESSID"。

要使用预定义常量 SID，通常要满足以下两个条件。

（1）处于会话活动状态，即已经调用 session_start()启动或回复会话。只有处于会话活动状态，才能获取会话 ID，否则 SID 是无定义的。

（2）客户端浏览器不支持 Cookie 机制（或者被用户禁用），或者配置文件 php.ini 中的 session.use_cookies 项被设置为 0。如果当前请求携带一个包含会话 ID 的 Cookie，那么 SID 将为空串。

9.2.3 会话变量

前面已经介绍和使用过系统预定义的超全局数组$_SESSION。这个数组用于保存会话数据，在会话伊始是空的。该数组中的每一个元素称为一个会话变量，元素的键就是变量名，元素的值就是变量值。在会话期间，可以往该数组添加数组元素，通常称之为注册会话变量。会话变量由 PHP 系统维护，在整个会话期间都是有效的。例如：

```
$_SESSION["name"] = $name;      // 注册一个名为 name 的会话变量
echo $_SESSION["name"];         // 访问一个名为 name 的会话变量
```

需要时，也可以调用 session_destroy 函数清除会话变量。

```
bool session_destroy()
```

该函数可以清除当前会话中所有注册的会话变量，但不会重置与会话相关的 Cookie。因为会话 ID 通常通过 Cookie 来保存的，因此该函数并不结束一次会话，即会话 ID 并没有丢失或改变。

清除会话变量发生在当前页面的所有 PHP 脚本代码被执行之后，所以在调用该函数之后、脚本代码结束执行之前，仍然可以访问所需的会话变量。

【例 9-3】 重写 URL 与使用会话变量。单击页面中的超链接可以再次访问此页面，页面显示该用户访问此页面的次数。代码如下：

```
<?php
session_start();
if (!array_key_exists('count', $_SESSION)) {
    $_SESSION['count'] = 1;
} else {
    $_SESSION['count']++;
}
?>
<p>
    嗨，你已访问该页面<?php echo $_SESSION['count']; ?>次.
</p>
<p>
    <a href="9-3.php?<?php echo SID; ?>">点击继续</a>.
</p>
```

无论配置文件（php.ini）中的 session.use_cookies 是否设置为 1，也不管浏览器是否支持 Cookie，该页面都是有效的。

9.3 页面跳转与重定向

在万维网上，网页之间通常会通过超链接等技术连接在一起，用户可以从一个网页跳转到另一个网页，从一个网站进入到另一个网站。在一个 Web 应用中，网页资源之间的连接程度就更加紧密了。下面介绍在 PHP 中实现网页跳转的几种常用技术。

一是使用超链接<a>元素。在页面中放置适当的<a>元素，当页面呈现后，用户单击相应的超链接文字，就可以跳转到指定的页面。

二是使用表单<form>元素。当用户单击表单的提交按钮后，不仅可以提交表单数据，还可以跳转到指定的页面。这里，指定的页面一般为动态页面，它既能处理用户提交的数据，又能把处理结果（如查询到的课程信息等）回送到客户端，呈现给用户。

三是使用特定的<meta>标签。<meta>标签用于定义页面的元数据，通常放置在页面头部。这里说的特定的<meta>标签，是指其 http-equiv 属性被设置为"refresh"，例如：

```
<meta http-equiv="refresh" content="5;url=process.php">
```

当包含此标签的页面在客户端浏览器上呈现后，浏览器会在 5s 后自动跳转到指定的页面 process.php，即用该指定页面的内容刷新当前页面。

最后是使用 PHP 中特定的 header 函数。这种方式有别于前面 3 种方式，它实现的页面跳转触发于服务器端。header 函数一般用于设置响应域。这里说的特定的 header 函数，是指其参数值的格式是特殊的，即在"Location:"后指定一个 URL，例如：

```
header("Location: response_1.php");
```

当服务器执行该函数时，会产生重定向响应。重定向响应一般只包含响应的状态行和响应头，没有响应体。其中响应的状态码为 302，表示重定向；响应头包括 Location 域，指定重定向的目标页面的 URL。

一般情况下，执行了这样的 header 函数后，就没有必要再执行后面的代码了。通常可以通过语言结构 exit 结束当前脚本代码的执行，直接产生重定向响应送往客户端。

当浏览器接收到重定向响应后，会自动向目标页面发出 GET 请求，这一过程并不需要用户的介入。最终，浏览器地址栏中显示目标页面的 URL，浏览器窗口中显示目标页面的内容。

上述 header 函数实现的页面跳转过程称为重定向（Redirect）。重定向经常与 POST 请求配合使用，产生一种 PRG（Post-Redirect-Get）模式，如图 9-4 所示。

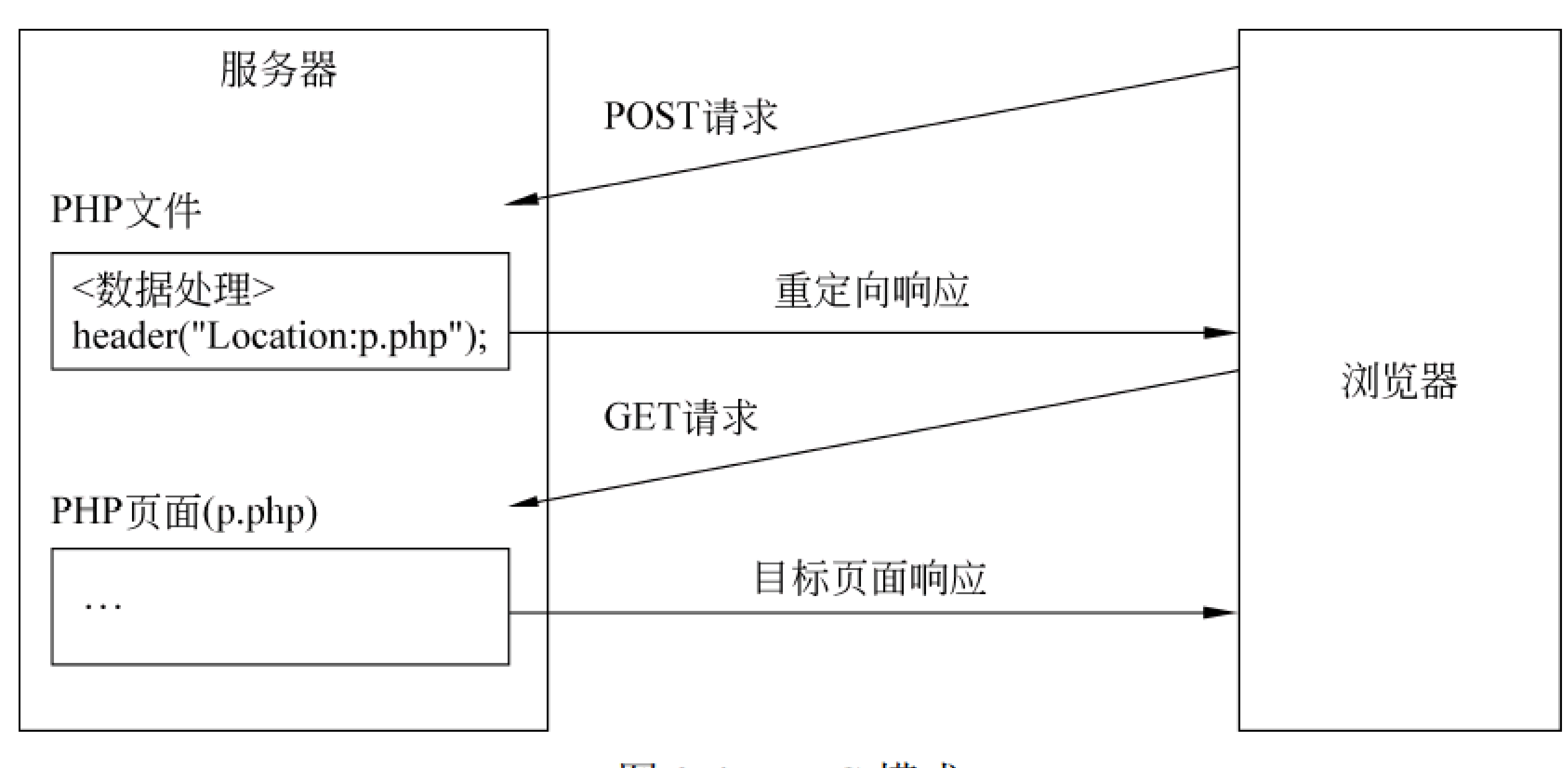


图 9-4　PRG 模式

在 PRG 模式中，当用户发送 POST 请求时，可以提交给某个 PHP 文件来处理，该 PHP 文件只负责数据的处理，而不负责内容的响应。当该 PHP 文件完成数据处理后，可以调用上述 header 函数，产生一个重定向响应。浏览器接收到重定向响应后，自动产生对目标页面的 GET 请求，最终由该目标页面产生响应内容。

【例 9-4】 修改例 9-2 中对表单数据的处理流程：如果所有的表单数据都通过检验，那么就把这些数据保存在会话变量中，然后重定向至另一个页面，该页面显示相关的会话数据。

该例包含两个 PHP 文件：一个是 9-4.php，是从 9-2.php 修改而来；另一个是 show.php，用于显示相关的会话数据。

下面是文件 9-4.php 的代码，其中与文件 9-2.php 相同的部分代码做了省略处理。文件 form2.php 的代码见例 9-2，没有改变。

```
<?php
// 定义变量并设置为空值
$user = $gender = $email = "";
$userErr = $genderErr = $emailErr = "";
$flag = true;

if($_SERVER["REQUEST_METHOD"] == "POST") {
    …              //（省略了）
}
if($_SERVER["REQUEST_METHOD"]=="POST" && $flag) {
    session_start();
    $_SESSION["user"] = $user;
    $_SESSION["gender"] = $gender;
    $_SESSION["email"] = $email;
    header("Location: show.php");
} else {
    include "form2.php";
}
?>
```

下面是文件 show.php 的代码。

```
<?php
    session_start();
    echo "用户名：", $_SESSION["user"], "<br />";
    echo "性别：", $_SESSION["gender"], "<br />";
    echo "email 地址：", $_SESSION["email"], "<br />";
?>
```

9.4　文件上传与下载

将文件从服务器下载到客户机，或把文件从客户机上传到服务器，是大多数 Web 应用都可能需要提供的功能。本节介绍在 PHP 中实现文件上传与下载的方法和技术。

9.4.1　文件操作

文件的上传和下载往往会涉及文件操作，例如读取文件内容、检测文件的大小等。这里介绍有关文件操作的一些基本函数。

1. 文件的打开和关闭

文件操作的一般过程是，先打开文件，然后对文件进行读写操作，最后关闭文件。

（1）fopen 函数。fopen 函数用于打开一个文件，以便对文件进行读写操作。该函数的

语法格式如下：

```
resource fopen($filename, $mode [, bool $use_include_path = false])
```

参数$filename 指定要打开文件的文件标识符。如果操作成功，函数返回一个指向文件的文件指针资源；如果文件不存在或没有该文件的访问权限，函数返回 false。

参数$mode 指定文件的打开模式，即将要对打开的文件做何种类型的操作。该参数的可取值及其含义见表 9-1。

表 9-1　文件打开模式

模式	含义	说　明
"r"	只读	从文件头开始读
"r+"	读写	从文件头开始读写
"w"	只写	打开并清除文件内容，若文件不存在就创建一个新文件
"w+"	读写	打开并清除文件内容，若文件不存在就创建一个新文件
"a"	只写	打开并在文件的末尾写，若文件不存在就创建一个新文件
"a+"	读写	在文件末尾写，若文件不存在就创建一个新文件
"x"	只写	创建一个新文件。若文件已经存在，产生警告信息，函数返回 false
"x+"	读写	创建一个新文件。若文件已经存在，产生警告信息，函数返回 false
"b"	二进制模式。用于连接在其他模式后面	

参数$use_include_path 是可选的，默认值为 false，此时 PHP 会在当前被请求的 PHP 文件所在的目录中寻找要打开的文件。如果将该参数设置为 true，那么 PHP 将在配置文件 php.ini 中的 include_path 项设置的路径中搜寻要打开的文件。

（2）fclose 函数。fclose 函数用于关闭一个已打开的文件，其语法格式如下：

```
bool fclose(resource $handle)
```

参数$handle 是一个已打开文件的文件指针，函数关闭该文件指针指定的文件。如果关闭成功，函数返回 true；否则函数返回 false。

2. 文件的写入

这里介绍文件写入的两个函数：fwrite 和 fputcsv。

（1）fwrite 函数。fwrite 函数用于向文件写入内容，其语法格式如下：

```
int fwrite(resource $handle, string $string [, int $length])
```

参数$handle 是一个已打开文件的文件指针，函数将向该文件指针所指文件写入数据。参数$string 指定要写入文件的字符串数据。参数$length 是一个可选项，如果指定，那么当写入了$string 中的前$length 个字节的数据后结束操作。如果$string 中的字节数小于$length，则在写入整个字符串$string 后结束操作。

如果写入操作成功，函数返回实际写入文件的字节数；否则函数返回 false。

（2）fputcsv 函数。fputcsv 函数用于向 CSV 文件写入一条记录。CSV（Comma-Separated Values）是一种通用的、相对简单的纯文本文件格式，其文件一般以.csv 作为扩展名。CSV

文件由记录组成（典型的是每行一条记录），每条记录由分隔符分隔为若干字段，每条记录都有相同的字段序列。

fputcsv 函数的语法格式如下：

```
int fputcsv(resource $handle, array $fields [, string $delimiter = ','
[, string $enclosure = '"']])
```

参数$handle 是一个已打开的 CSV 文件的文件指针，函数将向该文件指针所指文件写入一条记录。参数$fields 是一个数组，指定要写入文件的记录数据，每个数组元素对应一个字段。参数$delimiter 指定字段之间的分隔符（只能一个字符），默认值为逗号。参数$enclosure 指定字段值的定界符（只能一个字符），默认值为双引号。

如果写入操作成功，函数返回实际写入文件的字节数；否则函数返回 false。

3. 文件的读取

PHP 为文件的读取提供了许多函数，这里介绍几个最常用的。

（1）fread 函数。fread 函数用于读取文件的内容，其语法格式如下：

```
string fread(resource $handle, int $length)
```

参数$handle 是一个已打开文件的文件指针，函数将从该文件指针所指文件中读取数据。参数$length 指定要读取的最多字节数。当读取了指定的字节数或者碰到了文件尾时，读操作结束，函数返回读取的字符串。如果读操作出错，函数返回 false。

（2）fgets 函数。fgets 函数用于从文件中读取一行内容，其语法格式如下：

```
string fgets(resource $handle [, int $length])
```

参数$handle 是一个已打开文件的文件指针，函数将从该文件指针所指文件中读取数据。参数$length 是可选的，如果指定，那么函数最多读取$length–1 个字节。当读取了$length–1 个字节，或者读到了新行符（回车符、换行符或回车换行符），或者碰到了文件尾时，读操作结束，函数返回读取的字符串。如果没有任何数据可读（文件指针指向文件尾），或者读操作出错，函数返回 false。

（3）fgetcsv 函数。fgetcsv 函数用于从 CSV 文件中读取一条记录，其语法格式如下：

```
array fgetcsv(resource $handle [, int $length=0 [, string $delimiter=','
[, string $enclosure='"']]])
```

参数$handle 是一个已打开的 CSV 文件的文件指针，函数将从该文件指针所指文件中读取一条记录。参数$length 应该大于文件中最长一行的长度，如果忽略该参数，或将其设置为 0，则没有长度限制，该参数的默认值为 0。参数$delimiter 指定字段之间的分隔符（只能一个字符），默认值为逗号。参数$enclosure 指定字段值的定界符（只能一个字符），默认值为双引号。

函数读取指定 CSV 文件的当前记录，并解析出该记录的各字段值。函数返回一个包含这些字段值的数字索引数组。

如果参数$handle 无效，函数返回 NULL。如果没有任何数据可读（文件指针指向文件

尾），或者读操作出错，函数返回 false。

（4）feof 函数。feof 函数用于检测文件指针是否指向了文件尾，其语法格式如下：

```
bool feof(resource $handle)
```

参数$handle 是一个已打开文件的文件指针，若文件指针指向了文件尾，函数返回 true；否则函数返回 false。

（5）rewind 函数。rewind 函数将文件指针倒回到文件的开头，其语法格式如下：

```
bool rewind(resource $handle)
```

如果操作成功，函数返回 true；否则函数返回 false。

【例 9-5】 文件读写操作。下面 PHP 文件的功能是打开一个 CSV 文件（如果文件不存在则自动新建），然后往文件中添加两条记录，每条记录包含一本图书的 ISBN 号、书名、单价和出版社，最后读取该 CSV 文件中的所有图书信息并输出。代码如下：

```
<?php
$books = array(
    array("9877115255352", "计算机系统", "45.5", "电子工业出版社"),
    array("9847223255123", "PHP Web 应用开发", "35.2", "清华大学出版社")
);
$fp = fopen("files/aaa.doc", "a+");
foreach ($books as $value) {
    fputcsv($fp, $value);
}
rewind($fp);
while($book = fgetcsv($fp)) {
    list($isbn, $title, $price, $publisher) = $book;
    printf("<p>%s, %s, %6.2f, %s</p>", $isbn, $title, $price, $publisher);
}
fclose($fp);
echo "<p>文件操作结束!</p>";
?>
```

4. 其他函数

这里介绍 readfile、filesize 和 iconv 这 3 个函数。虽然它们都与文件操作有关，但不以文件指针作为参数，所以在调用前不需要先打开文件。

（1）readfile 函数。readfile 函数用于读取并输出一个文件的内容，其语法格式如下：

```
int readfile(string $filename [, bool $use_include_path = false])
```

参数$filename 指定文件标识符，函数读取指定文件的内容并将其写入到输出缓冲区。函数返回实际从文件中读入的字节数。若操作出错，函数返回 false。

想要在 include_path 中搜索文件，可将可选的第 2 个参数$use_include_path 设置为 true。readfile 函数既完成了读取文件内容的操作，又实现了文件内容的输出功能。

（2）filesize 函数。filesize 函数用于获取指定文件的大小，其语法格式如下：

```
int filesize(string $filename)
```

参数$filename 指定文件标识符，函数返回指定文件的大小，即文件的字节数。若操作出错，函数返回 false。

（3）iconv 函数。在上述函数中，很多都含有文件标识符（文件名）参数，用于指定要对哪个文件进行操作。如果文件名采用的编码不合适，就可能找不到指定的文件。例如，页面中的文件名采用 UTF-8 字符集编码，而文件系统中的文件名采用 GBK 字符集编码，那么就会导致文件定位失败。所以在进行文件操作前，可以先用 iconv 函数对文件名的编码做适当的转换。该函数的语法格式如下：

```
string iconv(string $in_charset, string $out_charset, string $str)
```

函数将字符串$str 从$in_charset 字符集编码转换为$out_charset 字符集编码。

9.4.2 文件上传

文件上传一般涉及两部分代码的设计：一是呈现于客户端的包含文件域在内的 HTML 表单；二是运行于服务器端的用于接收包括上传文件内容在内的表单数据的代码。

1. 文件上传表单

要实现文件上传，需要在 HTML 表单中包含 type 属性值为"file"的<input>元素，即文件域。在浏览器中，文件域一般呈现为一个按钮和一个框，按钮的标签可能会因浏览器不同而不同，如“浏览”“选择文件”等。当用户单击按钮时，会打开一个对话框供用户从本地文件系统中选择文件，被选择的文件的标识符会显示在框中。

对于大多数表单，<form>元素的 method 属性值可以取"GET"或"POST"，enctype 属性值通常取默认值，即"application/x-www-form-urlencoded"。而对于文件上传表单，method 属性值应该取"POST"，enctype 属性值必须取"multipart/form-data"，如下面代码：

```
<form enctype="multipart/form-data" method="POST" action="…">
  <其他控件元素或文件域>…
  <input id = "myfile" type = "file" name = "upfile" />
  <input type = "submit" value = "submit" />
</form>
```

在文件上传表单中，可以包含一个或多个文件域。除了文件域，也可以包含其他的表单控件元素。

2. 获取上传文件

在 PHP 中，可以很容易获取上传文件和其他表单数据。获取其他表单数据的方法跟之前介绍的没有区别，可以通过访问$_POST 数组获得。

要获得上传文件，先要知悉上传文件的有关属性。可以通过访问超全局变量$_FILES 来获得上传文件的有关属性。$_FILES 是一个关联二维数组，其外层数组的每个元素值是一个数组，表示一个上传文件，键是对应的文件域控件的名称。例如'upfile'；内层数组的每个元素值是该上传文件的某个属性，键是预定义的，例如'error'、'tmp_name'等。

- $_FILES['upfile']['error']：错误信息代码。值为 0 表示上传成功；值为 1 表示上传文件大小超过了配置文件 php.ini 中 upload_max_filesize 项指定的限制值；值为 2 表示上传文件大小超过了 HTML 表单中规定的最大值；值为 3 表示文件只有部分被上传；值为 4 表示没有文件被上传。
- $_FILES['upfile']['name']：上传文件的原文件名。
- $_FILES['upfile']['type']：上传文件的类型（MIME）。
- $_FILES['upfile']['size']：上传文件的大小，单位为字节。
- $_FILES['upfile']['tmp_name']：文件上传后在服务器端储存的临时文件的文件标识符。

PHP 在处理上传文件表单时，会自动把上传文件保存在临时目录中，这个临时文件的文件标识符可以通过访问数组元素$_FILES['upfile']['tmp_name']获得。不管上传是否成功，PHP 脚本代码执行完后，PHP 都会删除这些临时文件。所以在处理上传文件时，都需要将这些上传的临时文件移动到其他位置，或者读取上传文件的内容做所需的处理。

PHP 的 move_uploaded_file 函数可以完成上传文件的移动，该函数的语法格式如下：

```
bool move_uploaded_file(string $filename, string $destination)
```

函数移动一个上传的文件至新的位置。$filename 指定要被移动的上传文件的文件标识符，$destination 指定文件移动后目标文件的文件标识符。若操作成功，函数返回 true。如果$filename 不是一个上传的文件或因其他原因导致操作失败，函数返回 false。

【例 9-6】 假设数据库 test 中有一个 book 表，包含图书的编号（ISBN）、书名（title）和封面图像（cover）等字段。请设计一个表单，如图 9-5 所示，可以上传一本图书的相关信息，并保存到 test 数据库的 book 表中。



图 9-5　上传图书信息表单

该例结果包含两个文件：一个是用于呈现表单的页面文件，文件名是 9-6.html；另一个是用于处理上传数据的 PHP 文件，文件名为 up_process.php。

下面是文件 9-6.html 的代码：

```
<!DOCTYPE html>
<html>
```

```
  <head>
    <title>上传图书封面</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="formone.css"/>
  </head>
  <body>
    <form enctype="multipart/form-data" method="POST" action="up_process.php">
    <div class="outer">
    <div class="inter">
      <p>
      <label for="i1" class="label">ISBN</label>
      <input type="text" id="i1" name="isbn" maxlength="13"
               style="width: 120px" />
      </p>
      <p>
      <label for="i2" class="label">书名</label>
      <input type="text" id="i2" name="title" maxlength="20"
               style="width: 300px" />
      </p>
      <p>
      <label for="i3" class="label">封面</label>
      <input type="file" id="i3" name="cover" style="width: 300px" />
      </p>
      <p style="text-align: center; padding-top: 10px">
      <input type="submit" name="submit" value="提交"/>
      </p>
    </div>
    </div>
    </form>
  </body>
</html>
```

表单的呈现用到了外部样式表文件 formone.css，该样式表文件已经在第 2 章的例 2-20 中提出和介绍，这里不再赘述。

下面是文件 up_process.php 的代码：

```
<?php
header("Content-type:text/html;charset=UTF-8");
$isbn = $_POST['isbn'];
$title = $_POST['title'];
if(!empty($isbn) && !empty($title) && $_FILES['cover']['error']===0) {
  $filespec = $_FILES['cover']['tmp_name'];
  /* 连接数据库 */
  @ $mysqli = new mysqli("127.0.0.1", "root", "", "test", 3306);
```

```
    if ($mysqli->connect_errno) {
       echo "不能连接到数据库<br/>";
       return;
    }
    $mysqli->query("SET NAMES 'utf8'");
    /* 打开文件，获取上传图像文件内容 */
    $fp = fopen($filespec, "r");
    $cover = "";
    while(!feof($fp)) {
       $cover .= fread($fp, 1024);
    }
    $cover = addslashes($cover);
    /* 确定该图书编号是否已经存在 */
    $query = "SELECT * FROM book WHERE isbn='$isbn'";
    $result = $mysqli->query($query);
    if($result->num_rows>=1) {
       echo "该图书编号已经存在！";
       return;
    }
    /* 往数据库插入上传数据 */
    $sql = "INSERT INTO book VALUES('$isbn', '$title', '$cover')";
    $ret = $mysqli->query($sql);
    if(!$ret) {
       echo "数据保存操作失败！";
       return;
    }
    $mysqli->close();
    echo "数据已成功上传并保存！";
} else {
    echo "数据为空，或文件上传失败！";
}
?>
```

9.4.3 文件下载

在 Web 应用中，文件下载通常采用 HTTP 协议，即服务器以 HTTP 响应的形式向客户端发送文件，其中响应体就是被下载文件的内容。

实际上，当通过超链接元素<a>请求一个 HTML 文档时，就涉及对该文档的下载；当页面中通过<img>元素包含一个图像时，也涉及对该图像文件的下载。只不过，对这两种类型的文件，浏览器在接收时一般会直接打开呈现，所以主要用于浏览。

除了 HTML 文档和图像文件，也可以用<a>元素来请求其他类型的文件，例如：

```
<a href="files/ttt.txt">打开文件 ttt.txt </a>
<a href="files/logo.jpg">打开文件 logo.jpg </a>
<a href="files/chapter1.ppt">打开文件 chapter1.ppt </a>
```

当浏览器接收到 HTTP 响应后，如果响应内容是文本文件、图像文件等，就会直接打开呈现；如果是浏览器本身无法解析处理的文件，那么可能会直接下载保存，也可能会借助其他的应用软件打开。

这种用<a>元素来下载文件方式的优点是简单，即开发人员只需在元素的 href 属性中指定要下载的文件，当用户单击超链接文字发出 GET 请求后，服务器就会自动读取文件内容作为响应体，同时也会自动设置响应头中相应的域，如设置 Content-Type 域，指定文件的 MIME 类型；设置 Content-Length 域，指定文件的字节数等。

但这种方式也有其缺点，具体如下：

（1）把文件在服务器端的路径直接暴露给了用户，可能会有安全隐患；

（2）不适用文件内容保存在数据库里等情况。

更为一般的文件下载方法是由服务器端代码控制文件的下载，即由代码设置响应。通常，为实现文件下载，代码需要完成以下工作：

（1）设置响应域 Content-Type，指定文件的 MIME 类型。

（2）设置响应域 Content-Length，指定文件的大小，也即响应体包含的字节数。

（3）设置响应域 Content-Disposition，指定处置响应体内容的方式。

（4）读取文件内容，并将其作为响应体输出。通常，只需用 echo 或 print 等语言结构将整个文件的内容输出即可。

在下载文件时，通常要求浏览器以附件形式处置响应内容，并可以为之指定一个默认文件名。这是通过设置响应域 Content-Disposition 来实现的，例如：

```
header("Content-Disposition: attachment; filename=fn.doc");
```

这样，当浏览器接收到响应后，就不会试图去打开、呈现响应体内容，而会考虑将响应体内容作为文件去保存。比如，打开一个对话框，让用户选择保存文件的位置，而在响应域中指定的文件名（如 fn.doc）则会作为默认的文件名。

比较用<a>元素下载文件和用 PHP 代码下载文件两种方式。用<a>元素下载文件时，会自动设置 Content-Type 和 Content-Length 响应域，但不会设置 Content-Disposition 响应域，其主要作用是下载文件以呈现。用 PHP 代码下载文件时，可以设置 Content-Disposition 响应域，实现下载文件以保存的目的。但同时也要手动设置 Content-Type 和 Content-Length 响应域，其中在设置 Content-Type 响应域时需要指定文件的 MIME 类型。表 9-2 列出了常用文件的扩展名与 MIME（Multipurpose Internet Mail Extensions，多用途互联网邮件扩展）类型的对应关系。

表 9-2　常用文件的扩展名与 MIME 类型的对应关系

扩展名	MIME 类型
.html	text/html
.txt	text/plain
.gif	image/gif
.jpeg,.jpg	image/jpeg

续表

扩展名	MIME 类型
.png	image/png
.doc	application/msword
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document
.xls	application/vnd.ms-excel
.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
.ppt	application/vnd.ms-powerpoint
.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
.pdf	application/pdf
.zip	application/zip
.rar	application/x-rar-compressed
任意的二进制	application/octet-stream

【例 9-7】 设计一个页面，可以显示指定图书的 ISBN 号和其封面图像。ISBN 号是指定的，封面图像文件的内容保存在 test 数据库的 book 表中。通过 ISBN 号可以获取相应图书的封面图像内容。

该例结果包含两个文件：一个是作为页面的 PHP 文件，文件名为 9-7.php；另一个是用于读取并下载封面图像内容的 PHP 文件，文件名为 cover.php。

下面是文件 9-7.php 的代码：

```
<!DOCTYPE html>
<?php
$isbn = "9787302309796";
echo "书号：", $isbn, "<br />";
?>
<img src="cover.php?isbn=<?php echo $isbn;?>">
```

下面是文件 cover.php 的代码：

```
<?php
$isbn = $_GET["isbn"];
/* 连接数据库 */
@ $mysqli = new mysqli("127.0.0.1", "root", "", "test", 3306);
if ($mysqli->connect_errno) {
    echo "不能连接到数据库<br/>";
    return;
}

/* 根据图书编号查询图书的封面图像数据 */
$mysqli->query("SET NAMES 'utf8'");
$query = "SELECT cover FROM book WHERE isbn='$isbn'";
$result = $mysqli->query($query);
if(!$result) {
```

```
    echo "SQL 语句执行失败！";
    return;
}
$row = $result->fetch_array();
$cover = $row['cover'];                      // 获取封面图像内容
$result->free();
$mysqli->close();

header("Content-Type: image/png");           // 设置响应域 Content-Type
header("Content-Length: ".strlen($cover));   // 设置响应域 Content-Length
echo $cover;                                 // 产生响应体
?>
```

这里假设保存在数据库中的封面图像的文件类型是 png。另外，由于下载的封面图像只需要在页面中呈现，所以不需要设置 Content-Disposition 响应域。

【例 9-8】 用 PHP 代码实现文件下载。页面文件 9-8.php 包含一个超链接，单击超链接文字将请求另一个 PHP 文件 downfile.php，并携带请求参数 filename。该请求参数指定要下载文件的文件名。downfile.php 文件产生一个 HTTP 响应，实现对指定文件的下载。

下面是文件 9-8.php 的代码：

```
<!DOCTYPE html>
<?php
header("Content-type:text/html;charset=UTF-8");
$fn = urlencode("课件 1.ppt");
echo "<a href='downfile.php?filename=$fn'>下载文件</a>";
?>
```

下面是文件 downfile.php 的代码：

```
<?php
$fn = $_GET["filename"];
$filename = iconv("utf-8", "GBK", $fn);      // 转换文件名的字符编码
$path = "files/$filename";              // 假设文件存放在 files 子文件夹下
$extname = getExtension($filename);          // 获得文件名的扩展名
$mime = getMime($extname);                   // 获得文件 MIME 类型

/* 设置响应域 Content-Length、Content-Type 和 Content-Disposition */
header("Content-Length: ".filesize($path));
header("Content-Type: $mime");
header("Content-Disposition: attachment; filename=$filename");
readfile($path);                             // 将文件内容作为响应体

/* 根据文件名获取文件的扩展名 */
function getExtension($filename){
    return preg_replace('/.+\\./', '', $filename);
```

```
}
/* 根据扩展名获取文件的 MIME 类型 */
function getMime($extname) {
    $mime = array();
    $mime["html"] = "text/html";
    $mime["txt"] = "text/plain";
    $mime["jpg1"] = "image/jpeg";
    $mime["doc"] = "application/msword";
    $mime["xls"] = "application/vnd.ms-excel";
    $mime["ppt"] = "application/vnd.ms-powerpoint";
    if(array_key_exists($extname, $mime)) {
        return $mime["$extname"];
    } else {
        return "application/octet-stream";
    }
}
?>
```

9.5　实例：管理员登录与退出

在本书 4.5 节的实例中，创建了管理员子系统的主页，但负责页面主区的模块文件 login_admin.php 还未实现。本实例将创建该模块文件，其功能是呈现登录表单并完成对登录数据的处理。包含该模块文件的子系统主页的呈现效果如图 9-6 所示。



图 9-6　主页（管理员登录）

在管理员子系统中，主页即为登录页面，也就是说必须先登录才能使用管理员子系统。进入子系统后，各功能页面都会包含子系统的导航栏，导航栏模块文件创建详见 3.4 节。导

航栏的右侧是菜单项“退出”，本实例将给出实现该菜单项功能的 PHP 文件 exit.php。

在本实例的最后，还会根据当前登录信息对本书 8.5 节实例中创建的“浏览教师信息”页面文件 teacher_p.php 进行完善。

模块文件 login_admin.php 以例 2-20 创建的表单文件 2-20.html 为基础，经过相应的修改并增加登录数据处理功能而形成。这里将该模块文件保存在教务选课系统项目 xk 的“源文件”结点下的 admin 子目录中。下面是该模块文件的代码：

```
<!--
*  功能：呈现登录表单、处理登录。登录成功时，将重定向至“浏览教师信息”页面
*  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
-->
<?php
include_once '../lib/mysql.php';
session_start();    // 启动或恢复会话
// 定义变量并设置为空值
$userErr = $pwErr = "";
$user = $pw = "";
/* 处理登入 */
if($_SERVER["REQUEST_METHOD"] == "POST" && array_key_exists("submit", $_POST)) {
   $flag = true;
   if(array_key_exists("user", $_POST)) $user = trim($_POST["user"]);
   if (empty($user)) {
      $userErr = "用户名是必填项";
      $flag = false;
   }
   if(array_key_exists("pw", $_POST)) $pw = trim($_POST["pw"]);
   if (empty($pw)) {
      $pwErr = "密码是必填项";
      $flag = false;
   }
   if($flag) {
      $mysqli = connect();
      if ($mysqli->connect_errno) {
         echo "不能连接到数据库<br/>";
         echo $mysqli->connect_error;
         exit();
      }
      $sql = "select * from teacher where tn = '$user' and admin = '是'";
      $result = executeSql($mysqli, $sql);
      if($result->num_rows === 1) {
         $row = $result->fetch_array();
         $pass = $row["tpassword"];
```

```
      if($pw === $pass) { // 登录成功，注册会话变量
        $_SESSION["user"] = $user;
        $_SESSION["name"] = $row["tname"];
        $_SESSION['dept'] = $row['dept'];
        $_SESSION["lb"] = '管理员';
        $mysqli->close();
        header("Location: teacher_p.php");
        exit();
      }
      $pwErr = '密码错误';
    } else {
      $userErr = '用户名错误，或不是管理员';
    }
    $mysqli->close();
  }
}
?>
<!-- 呈现表单 -->
<form class="style1" method="POST">
<div class="outer">
  <div class="title">管理员登录</div>
  <div class="inter">
    <p>
    <label for="i1" class="label">用户名</label>
    <input type="text" id="i1" name="user" maxlength="4"
           style="width: 60px" value="<?php echo $user ?>" />
    <span class="errMsg"><?php echo $userErr ?></span>
    </p>
    <p>
    <label for="i2" class="label">密  码</label>
    <input type="password" id="i2" name="pw" maxlength="12"
           style="width: 130px" value="<?php echo $pw ?>" />
    <span class="errMsg"><?php echo $pwErr ?></span>
    </p>
    <p style="text-align: center; padding-top: 10px">
    <input type="submit" class="big" name="submit" value="确  认"/>
    </p>
  </div>
</div>
</form>
```

该模块文件同样要用到在例 2-20 中定义的、保存于外部样式表文件 formone.css 的“表单样式”。这里只需把“表单样式”的所有 CSS 规则复制到本项目的外部样式表文件 xk.css 中即可，该文件处于“源文件”结点下的 css 子目录。

该模块文件运行时，会对登录数据进行处理。如果登录失败，会重新显示当前主页（登

录表单），并显示相应的错误信息。如果登录成功，则会将登录用户的相关信息注册为会话变量，然后重定向至“浏览教师信息”页面。

进入管理员子系统后，其他各功能页面都会呈现子系统的导航栏。单击导航栏中的菜单项“退出”将调用 PHP 文件 exit.php。下面是该 PHP 文件的代码：

```
<?php
session_start();    // 恢复会话
session_destroy();
header("Location: index_admin.php");
exit();
?>
```

注册了包含有登录管理员信息的会话变量后，就可以对“浏览教师信息”页面文件做相应的改进。

（1）在文件开头先判断是否有管理员登录，如果没有，就重定向至子系统主页，让用户先登录；

（2）在调用导航栏模块前，将输入变量$name 设置为当前登录管理员的姓名。下面是经过改进后的“浏览教师信息”页面文件 teacher_p.php 的代码，其中粗体部分是修改或添加的代码：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>浏览教师信息</title>
    <link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
  </head>
  <body>
    <?php
      session_start();
      $lb = @$_SESSION['lb'];
      if(empty($lb) || $lb !== '管理员') {
        header("Location: index_admin.php");
        exit();
      }
    ?>
    <?php include("header_admin.php"); ?>

    <?php
      $name = @$_SESSION['name'];
      $choice = 1;
      include("navigation_admin.php");
    ?>

```

```
        <div style="width: 90%; min-height: 150px; margin: 0 auto; padding: 5px;">
            <?php
                include("teacher_list.php");
            ?>
        </div>

        <?php include("footer_admin.php"); ?>
    </body>
</html>
```

最后更改管理员子系统的主页文件 index_admin.php，使页面的主区部分由登录模块文件 login_admin.php 负责呈现。

9.6 实例：添加课程

本实例实现管理员子系统中“添加课程”菜单项的功能。当单击导航栏中“添加课程”菜单项时，将请求“添加课程”页面文件 course_p.php。该页面包含两个视图，页面文件通过请求参数 v 的值决定呈现哪个视图。

第 1 个视图（v=1）是“添加课程”，对应模块文件 course_input.php，可以呈现添加课程的表单、实现为管理员所在部门添加新课程的功能，其呈现效果如图 9-7 所示页面的主区部分。



图 9-7 “添加课程”页面的“添加课程”视图

第 2 个视图（v=2）是“课程列表”，对应模块文件 course_list.php，可以分页呈现管理员所在部门负责的课程信息列表，其呈现效果如图 9-8 所示页面的主区部分。



图 9-8 “添加课程”页面的“课程列表”视图

给出“添加课程”页面文件 course_p.php 的代码。该文件利用 PHP 包含文件技术，通过包含页面头模块文件、导航栏模块文件、“添加课程”或“课程列表”模块文件以及页面脚模块文件组成完整的页面。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>添加课程</title>
    <link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
  </head>
  <body>
    <?php
      session_start();
      $lb = @$_SESSION['lb'];
      if(empty($lb) || $lb !== '管理员') {
        header("Location: index_admin.php");
        exit();
      }
      $name = @$_SESSION['name'];
      $dept = @$_SESSION['dept'];
    ?>
    <?php include("header_admin.php"); ?>
    <?php
      $choice = 2;
      include("navigation_admin.php");
```

```
        ?>
        <div style="width: 90%; min-height: 150px; margin: 0 auto; padding: 5px;">
            <?php
                $v = @$_GET['v'];
                if(empty($v)) $v = "1";
                if($v === "1") {
                    include("course_input.php");
                } else {
                    include("course_list.php");
                }
            ?>
        </div>
        <?php include("footer_admin.php"); ?>
    </body>
</html>
```

给出“添加课程”模块文件 course_input.php 的代码。该模块文件呈现于页面主区，是“添加课程”页面的第 1 个视图，在请求参数 v 等于 1 时被调用。其中，表单中的“负责教师”选择列表只包含管理员所属部门的教师，也就是说，管理员只能添加所属部门的新课程。这里，管理员只输入课程的课程号、课程名、学分和负责教师等信息，课程描述和课程大纲则由课程的负责教师在之后补充输入。代码如下：

```
<!--
*  功能：管理员向数据库添加所属部门的新课程（不包括课程描述和大纲）
*  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
*        $dept：登录管理员的所属部门，如"信息学院"
-->
<?php
include_once '../lib/mysql.php';
/* 连接数据库 */
$mysqli = connect();
if ($mysqli->connect_errno) {
    echo "不能连接到数据库<br/>";
    exit();
}
/* 初始化变量，这些变量的值将作为表单控件元素的值，或者是错误信息和提示信息 */
$cn = $cname = $credit = $tn = "";
$cnErr = $cnameErr = $creditErr = "";
$title = "请输入...";
/* 处理表单数据 */
if($_SERVER["REQUEST_METHOD"]==="POST" && array_key_exists("submit", $_POST)) {
    $flag = true;
```

```php
    if(array_key_exists("cn", $_POST)) $cn = trim($_POST["cn"]);
    if (empty($cn)) {
        $cnErr = "课程号是必填项";
        $flag = false;
    }
    if(array_key_exists("cname", $_POST)) $cname = trim($_POST["cname"]);
    if (empty($cname)) {
        $cnameErr = "课程名是必填项";
        $flag = false;
    }
    if(array_key_exists("credit", $_POST)) $credit = trim($_POST ["credit"]);
    if (empty($credit)) {
        $creditErr = "学分是必填项";
        $flag = false;
    } else {
        $pattern = '/^\d$/';
        if(preg_match($pattern, $credit) != 1) {
            $creditErr = "学分值只能取 1-9";
            $flag = false;
        }
    }
    $tn = trim($_POST["tn"]);
    if($flag) {
        $sql = "select * from course where cn = '$cn'";
        $result = executeSql($mysqli, $sql);
        if($result->num_rows === 1) {
            $cnErr = "课程号已存在";
            $flag = false;
        }
    }
    if($flag) {  // 数据有效，向数据库添加课程信息
        $sql = "insert into course(cn, cname, credit, tn) values('$cn', '$cname', $credit, '$tn')";
        $result = executeSql($mysqli, $sql);
        $title = "已成功添加课程，请继续...";
        $cn = $cname = $credit = $tn = "";    // 初始化表单控件值
    } else {
        $title = "数据有错，请修改...";
    }
}
?>
<?php
// 根据当前管理员所属部门，构建表单中“负责教师”选择列表中各选项的 HTML 代码
$sql = "SELECT tn, tname FROM teacher WHERE dept = '$dept'";
$result = executeSql($mysqli, $sql);
```

```
$options = "";
while($row = $result->fetch_array()) {
  $options .= "<option value='".$row['tn']."'";
  $options .= $tn===$row['tn']?" selected='selected'>" : ">";
  $options .= $row['tname']."</option>\r\n";
}
$mysqli->close();
?>
<!-- 呈现课程信息录入表单 -->
<div style="margin: 20px 0 30px 0">
  如果要查看课程信息，请单击<a href="course_p.php?v=2">课程列表</a>
</div>
<div class="title" style="margin-bottom: 20px"><?php echo $title ?>
</div>
<form method="POST">
  <p>
    <label for="i1" class="label">课程号</label>
    <input type="text" id="i1" name="cn" maxlength="10"
           style="width: 100px" value="<?php echo $cn ?>" />
    <span class="errMsg"><?php echo $cnErr ?></span>
  </p>
  <p>
    <label for="i2" class="label">课程名</label>
    <input type="text" id="i2" name="cname" maxlength="20"
           style="width: 300px" value="<?php echo $cname ?>" />
    <span class="errMsg"><?php echo $cnameErr ?></span>
  </p>
  <p>
    <label for="i3" class="label">学分</label>
    <input type="text" id="i3" name="credit" maxlength="1"
           style="width: 30px" value="<?php echo $credit ?>" />
    <span class="errMsg"><?php echo $creditErr ?></span>
  </p>
  <p>
    <label for="i4" class="label">负责教师</label>
    <select id="i4" name="tn"><?php echo $options; ?></select>
  </p>
  <p style="padding-top: 10px">
    <input type="submit" class="big" name="submit" value="确 认"/>
  </p>
</form>
```

最后给出“课程列表”模块文件 course_list.php 的代码。该模块文件呈现于页面主区，是“添加课程”页面的第 2 个视图，在请求参数 v 等于 2 时被调用。代码如下：

```
<!--
```

```
 *  功能：分页呈现登录管理员所属部门的课程信息列表
 *  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
 *        $dept：登录管理员的所属部门，如"信息学院"
-->
<?php
include_once '../lib/mysql.php';
include_once '../lib/pager.php';
/* 连接数据库 */
$mysqli = connect();
if ($mysqli->connect_errno) {
   echo "不能连接到数据库<br/>";
   return;
}
/* 通过查询数据库获取总行数，设置页面大小，并计算总页数 */
$query = "SELECT COUNT(*) FROM course c, teacher t WHERE c.tn=t.tn and t.dept='$dept'";
$result = executeSql($mysqli,$query);
$rows = $result->fetch_array()[0];
$pageSize = 3;                               // 设置页面大小
$pageCount = (int)ceil($rows/$pageSize);    // 总页数
/* 确定当前页码 */
@ $currentPage = $_GET['p'];
if(empty($currentPage)) $currentPage = 1;
elseif($currentPage<1) $currentPage = 1;
elseif($currentPage>$pageCount) $currentPage = $pageCount;
/* 构建翻页导航栏的 HTML 代码或者是相当的信息，并保存在变量$links 中 */
$links = "";
if($rows === 0) {
   $links = "<span>无数据！</span>";
} elseif($pageCount === 1) {
   $links = "<span>共{$rows}条记录！</span>";
} else {
   $showPages = 1;  // 设置翻页导航栏中连续页码超链接数
   list($startPage, $endPage) = getBounds($pageCount, $currentPage, $showPages);
   $url = $_SERVER['SCRIPT_NAME']."?v=2";
   $links = getLinks($startPage, $endPage, $pageCount, $currentPage, $url);
}
/* 获取当前页的数据记录 */
$first = ($currentPage-1)*$pageSize;
$query = "SELECT cn, cname, credit, tname FROM course c, teacher t "
       . "WHERE c.tn=t.tn and t.dept = '$dept' ORDER BY cn LIMIT $first, $pageSize";
$result = executeSql($mysqli,$query);
?>
<!-- 呈现当前页课程信息 -->
```

```
<style type="text/css">
  table .c1 {width: 120px; padding-left: 10px}
  table .c2 {width: 300px;}
  table .c3 {width: 60px;}
  table .c4 {width: 80px;}
</style>
<div style="margin: 20px 0 30px 0">
  如果要添加新课程，请单击<a href="course_p.php?v=1">添加课程</a>
</div>
<table>
  <thead>
    <tr>
    <td class="c1">课程号</td><td class="c2">课程名</td>
    <td class="c3">学分</td><td class="c4">教师</td>
    </tr>
  </thead>
  <tbody>
    <?php while($row = $result->fetch_array()) { ?>
      <tr>
      <td class="c1"><?php echo $row['cn'] ?></td>
      <td class="c2"><?php echo $row['cname'] ?></td>
      <td class="c3"><?php echo $row['credit'] ?></td>
      <td class="c4"><?php echo $row['tname'] ?></td>
      </tr>
    <?php } ?>
  </tbody>
  <tfoot>
    <tr style="height: 10px"><td colspan="4"></td></tr>
  </tfoot>
</table>
<div class="pager"><?php echo $links ?><div style="clear: both"></div></div>
<?php $mysqli->close() ?>
```

该模块文件利用函数库 pager.php 中的函数构建翻页导航栏的 HTML 代码。这里，在调用 getLinks 函数时传入第 5 个参数的值必须是“/xk/admin/course_p.php?v=2”，以确保翻页时仍请求该页面的第 2 个视图。getLinks 函数处理后产生的各页码超链接的 href 属性值会在此基础上再增加有关页码的请求参数，例如 “/xk/admin/course_p.php?v=2&p=2”。

习 题 9

1. 根据要求写 PHP 代码

（1）重定向至页面文件 courses.php。

（2）已知$f_name 是一个使用 utf-8 编码的字符串，请将其转换成使用 gbk 编码的字符串。

（3）假设上传文件域控件的 html 代码如下：

```
<input id="myfile" type="file" name="upfile" style="width: 300px; "/>
```

请使用 PHP 代码获取上传文件保存在服务器端的临时文件的文件标识符，并保存在变量$o_name 中。

（4）假设已通过下面代码获取文件上传后在服务器端储存的临时文件的文件标识符：

```
$o_name = $_FILES['upfile'][ 'tmp_name'];
```

请使用 PHP 代码将该上传文件移至新的位置。$d_name 是目标文件的文件标识符。

2. 简答题

（1）有 page1.php 和 page2.php 两个页面，代码如下：

```
page1.php:
<form action="page2.php">
    Please Enter:
    <input id="id1" name="c1" type="text" />
    <input id="id2" name="c2" type="submit" />
</form>
page2.php:
<?php
$content1 = $_POST['c1'];
echo "content1:$content1";
?>
```

现在 page1.html 页面已经打开，如果在其中的文本域中输入：true，然后单击提交按钮，那么 page2.php 文件将被执行。试问代码执行时是否会产生错误信息？echo 语句的输出结果是什么？请说明原因。

（2）有 page1.php 和 page2.php 两个页面，代码如下：

```
page1.php:
<form action="page2.php">
   Please Enter:
   <input id="id1" name="c1" type="text" />
   <input id="id2" name="c2" type="submit" />
</form>
page2.php:
<?php
$content1 = $_GET['id1'];
echo "content1:$content1";
?>
```

现在 page1.html 页面已经打开，如果在其中的文本域中输入：true，然后单击提交按钮，那么 page2.php 文件将被执行。试问代码执行时是否会产生错误信息？echo 语句的输出结果是什么？请说明原因。

（3）有 page1.php 和 page2.php 两个页面，代码如下：

```
page1.php:
<form action="page2.php">
   Please Enter:
   <input id="id1" name="c1" type="radio" value="男" />男
   <input id="id2" name="c1" type="radio" value="女" />女
   <input id="id3" type="submit" value="OK" />
</form>
page2.php:
<?php
$content1 = $_GET['c1'];
echo "content1:$content1";
?>
```

现在 page1.html 页面已经打开，如果没有选择任何一个单选按钮，直接单击提交按钮请求 page2.php。试问 page2.php 代码执行时是否会产生错误信息？echo 语句的输出结果是什么？请说明原因。

（4）假设 PHP 应用 myweb 中有 page1.php 页面文件，代码如下：

```
<?php
function process($c, $d = 25) {
   $retval = $c + $d - $_GET['c'] - $e;
   return $retval;
}
$e = 10;
echo process(5);

?>
```

现在在浏览器地址栏中输入以下 URL 请求 page1.php 页面：

```
http://localhost/myweb/page1.php?c=25
```

试问代码执行时是否会产生错误信息？echo 语句的输出结果是什么？请说明原因。

（5）假设 PHP 应用 myweb 中有 page1.php 和 page2.php 两个页面，代码如下：

```
page1.php:
<?php
header("Location: page2.php");
echo "hello";
?>
page2.php:
<?php
```

```
echo "welcome";
?>
```

现在在浏览器地址栏中输入以下 URL 请求 page1.php 页面：

```
http://localhost/myweb/page1.php
```

那么，浏览器上的呈现结果是什么？浏览器地址栏中显示的 URL 是什么？请简述其请求-响应的过程。

第 10 章　使 用 数 组

本章主题：

- PHP 数组的概念;
- 创建和初始化数组;
- 操作数组元素;
- 遍历数组;
- 数组运算符;
- 数组排序;
- 数组的并集、交集和差集;
- 其他常用的数组函数。

在 PHP 中，数组是数据或元素的有序集合，每一个元素是一个键-值对。一个数组元素相当于一个简单变量，可以存储一个数据。可以通过数组的变量名及键来访问相应元素的值。数组为分类组织和存储数据以及有效处理数据提供了便捷的手段。

本章介绍数组的概念、数组创建、数组元素的操作以及数组的遍历等内容，同时也会介绍许多涉及数组操作的 PHP 内置函数。

10.1　什么是数组

在 PHP 中，数组是有序的映射。一个数组由若干元素组成，每个元素是一个键-值对。键用于索引数组元素，可以是整数也可以是字符串，一个数组各元素不能有重复的键。值也称为数组元素的值，可以是任意类型。

如果一个数组各元素的键采用从 0（或 1）开始连续的整数，那么这样的数组就类似于一般计算机语言中的数组。在 PHP 中，一个数组的元素的键不一定从 0（或 1）开始，也不一定是连续的，甚至整数和字符串可以混合使用，即有些元素的键是整数，有些元素的键是字符串。

有时候，把键为整数的数组称为数字索引数组，把键为字符串的数组称为关联数组。

数组元素的值可以是任意类型。如果一个数组所有元素的值都是标量类型的，如整数、浮点数、字符串等，那么这个数组通常被称为一维数组。如果一个数组的元素的值本身就是数组，就可形成多维数组。

在 PHP 中，不要求一个数组各元素的值具有相同的类型。比如有些元素的值是整数，其他一些元素的值是字符串；有些元素的值是标量类型的，其他一些元素的值是数组。

10.2 创建和初始化数组

通常，可以利用语言结构 array 创建数组，并指定数组的各初始元素。有时候也可使用函数 range 快速创建一个由指定范围内的值填充的数组。

10.2.1 使用 array 语言结构

语言结构 array 用于创建并初始化一个数组，其语法格式如下：

```
array([[<key>]=><value> [, [<key>]=><value>]*])
```

每个 key=>value 定义一个元素（键-值对），各元素之间用逗号分隔。最后一个元素的后面可以有逗号也可以没有。

指定一个元素时，键是可以缺省的。此时，PHP 将使用之前已经被使用的最大的整数键加 1 作为该元素的键。如果前面的元素没有使用过整数键，则该元素的键自动取 0。

一个数组各元素的键不能相同。如果指定了具有相同键的多个元素，则后面的元素将覆盖前面的元素。

【例 10-1】 创建数组。

```
<?php
/* 创建并初始化了一个包含 3 个元素数组，每个元素的键是水果名，值是水果的价格 */
$prices = array("apple" => 12.5, "orange" => 9.8, "banana" => 15.2);
print_r($prices);   // 输出:Array ( [apple] => 12.5 [orange] => 9.8 [banana] => 15.2 )
/* 创建并初始化了一个包含书名的数组 */
/* 第 1 个元素的键由 PHP 设为 0，第 3 个元素的键由 PHP 设置为 6 */
$books = array("数据库原理", 5=>"操作系统", "Java 教程");
print_r($books); // 输出: Array ( [0] => 数据库原理 [5] => 操作系统 [6] => Java 教程 )
?>
```

如果指定的键是以下类型的数据，PHP 将强制将其转换成整数。

（1）包含整数格式的字符串。比如键"8"被转换成 8。

（2）浮点数。丢弃小数部分，比如 8.7 被转换成 8。

（3）布尔值。true 转换成 1，false 转换成 0。

下面代码创建一个数组。

```
$array = array(2 => "a", "2.5" => "b", 2.5 => "c", "2" => "d");
print_r($array);     // 输出: Array ( [2] => d [2.5] => b )
```

其中，第 1、第 3 和第 4 个元素的键经自动转换后都是整数 2，后面的值覆盖前面的值，所以元素的最终值是"d"。第 2 个元素的键是字符串"2.5"，因为不是整数格式，所以不会被转换。数组最终包含两个元素，第 1 个元素的键是整数 2，第 2 个元素的键是字符串"2.5"。

一个数组元素的值可以是标量类型数据，也可以是数组本身，这样就能形成二维数组和多维数组。多维数组可以使用嵌套的 array 来创建。

【例 10-2】 创建二维数组。下面代码创建了一个二维数组，外层数组的每个元素表示一本图书，内层数组的各元素表示某图书的各属性，包括书名、单价和出版社。代码如下：

```
<?php
$books = array(array("数据库原理", 36.50, "高等教育出版社"),
               array("操作系统", 32.00, "清华大学出版社"),
               array("Java 程序设计", 35.00, "电子工业出版社")
 );
print_r($books);
?>
```

下面是代码运行的输出结果：

```
Array ( [0] => Array ( [0] => 数据库原理 [1] => 36.5 [2] => 高等教育出版社 )
        [1] => Array ( [0] => 操作系统 [1] => 32 [2] => 清华大学出版社 )
        [2] => Array ( [0] => Java 程序设计 [1] => 35 [2] => 电子工业出版社 )
)
```

10.2.2 使用 range 函数

range 函数可以快速创建一个由指定范围内的值填充的数组。其语法格式如下：

```
array range(mixed $start , mixed $end [, number $step])
```

其中，参数$start、$end 和$step 分别表示用于填充新创建数组的初值、最大值和步长。参数$step 是可选的，其默认值为 1。

【例 10-3】 使用 range 函数创建数组。代码如下：

```
<?php
$arr1 = range(1, 6);
print_r($arr1);  // 输出：Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 )
$arr2 = range(0, 58, 10);
print_r($arr2);  // 输出：Array ( [0] => 0 [1] => 10 [2] => 20 [3] => 30 [4] => 40 [5] => 50 )
$arr3 = range('a', 'f');
print_r($arr3);  // 输出：Array ( [0] => a [1] => b [2] => c [3] => d [4] => e [5] => f )
?>
```

10.3 操作数组元素

这里介绍对数组元素的操作，包括数组元素的访问、添加、修改和删除。另外也涉及用于测试数组元素是否存在的若干函数。

10.3.1　访问数组元素

可以通过方括号或花括号访问数组元素，其格式如下：

```
<array>[<key>]
```

或

```
<array>{<key>}
```

访问数组元素时，如果指定的键不存在，那么 PHP 系统将发出一个 Notice 错误信息，而访问结果是 NULL。

利用下面函数可以测试一个数组是否存在某个键和值。

（1）array_key_exists 函数。语法格式如下：

```
bool array_key_exists(mixed $key, array $array)
```

其功能是检测指定的$key 是否为数组$array 的某个元素的键，若是函数返回 true；否则函数返回 false。

（2）in_array 函数。语法格式如下：

```
bool in_array(mixed $needle, array $haystack [, bool $strict])
```

其功能是检测指定的$needle 是否为数组$haystack 的某个元素的值，若是函数返回 true，否则函数返回 false。

参数$strict 是可选的，默认值为 false，此时采用相等比较（==）。若将其设置为 true，则采用全等比较（===）。

（3）array_search 函数。语示格式如下：

```
mixed array_search(mixed $needle, array $haystack [, bool $strict])
```

其功能是检测指定的$needle 是否为数组$haystack 的某个元素的值，若是函数返回该元素的键，否则函数返回 false。若数组中存在多个元素都具有该值，则函数返回第一个元素的键。

【例 10-4】 访问和测试数组元素。下面代码涉及的数组是一个不规则的多维数组。外层数组包含 3 个元素，前两个元素的值是标量类型数据，第 3 个元素的值本身是一个二维数组。代码如下：

```
<?php
$arr = array(
      "x" => 5,
      5 => "x",
      "y" => array(
            "aa" => array(10 => "aaa"),
            "bb" => array(10 => "bbb")
       )
```

```
);
var_dump($arr["x"]);                // 输出: int(5)
var_dump($arr[5]);                  // 输出: string(1) "x"
var_dump($arr["y"]["aa"][10]);      // 输出: string(3) "aaa"
var_dump($arr["y"]["bb"][10]);      // 输出: string(3) "bbb"
echo "<br />";
/* 根据键检测元素 */
var_dump(array_key_exists(5, $arr));            // 输出: bool(true)
var_dump(array_key_exists("aa", $arr["y"]));    // 输出: bool(true)
echo "<br />";
// 根据值检测元素
var_dump(in_array("x", $arr));      // 输出: bool(true)
var_dump(array_search("x", $arr)); // 输出: int(5)
?>
```

10.3.2 修改、添加或删除数组元素

利用方括号（或花括号）既可以访问数组元素，也可以修改数组元素或添加数组元素，这只需要将值赋给指定的元素即可。

格式 1：

```
<array>[<key>]=<value>;
```

格式 2：

```
<array>[]=<value>;
```

格式 1 是将数组元素<array>[<key>]的值替换成<value>。如果数组中原先没有该元素，那么就添加这样一个元素。

格式 2 会在数组中添加一个元素，元素的值为<value>，键为数组之前已经被使用的最大的整数键加 1，或者为 0。

在修改或添加数组元素时，如果指定的数组并不存在，PHP 会自动创建数组。例如执行下面代码时，如果数组 new_arr 原先并不存在，PHP 会在执行第一条语句时先创建该数组，然后添加键为'x'、值为'aaaaa'的第一个元素。

```
$new_arr['x'] = 'aaaaa';
$new_arr['y'] = 'bbbbb';
$new_arr[] = 'ccccc';
```

说明：这种创建数组的方法是不提倡的。如果$new_arr 是一个事先存在的字符串变量，那么 PHP 会把方括号看作是字符串操作符。

相反，使用 unset 函数可以删除一个数组元素，甚至整个数组。例如：

```
unset($new_arr['y']);          // 删除数组 new_arr 中键为'y'的元素
unset($new_arr);               // 删除整个数组 new_arr
```

【例 10-5】 修改、添加或删除数组元素。代码如下：

```
<?php
$new_arr = array('x' => '11111');
$new_arr['x'] = 'aaaaa';
$new_arr['y'] = 'bbbbb';
$new_arr[] = 'ccccc';
print_r($new_arr);  // 输出：Array ( [x] => aaaaa [y] => bbbbb [0] => ccccc )
echo "<br />";
unset($new_arr["x"]);
print_r($new_arr);  // 输出：Array ( [y] => bbbbb [0] => ccccc )
echo "<br />";
unset($new_arr);
var_dump(isset($new_arr)); // 输出：bool(false)
?>
```

10.3.3 在数组头部或尾部操作元素

array_unshift 和 array_shift 函数可以分别在数组头部插入和删除元素，array_push 和 array_pop 函数可以分别在数组的尾部添加和删除元素。

1. array_unshift 函数

array_unshift 函数用于在参数数组$array 的头部插入一个或多个元素，其语法格式如下：

```
int array_unshift(array &$array, mixed $value [, mixed $value]*)
```

新插入元素的键是数值键，与数组中原先已存在的数值键一起，重新设置为从 0 开始的连续整数。数组中原有的字符串键不变。函数返回插入元素后数组的元素个数。

2. array_shift 函数

array_shift 函数用于从参数数组$array 的头部删除一个元素，其语法格式如下：

```
mixed array_shift(array &$array)
```

第一个元素被删除后，数组中其他数值键被重新设置为从 0 开始的连续整数。函数返回被删除元素的值。若数组原先是空的，函数返回 NULL。

【例 10-6】 在数组头部添加和删除元素。代码如下：

```
<?php
// 在头部添加元素
$arr = array("f" => 1, 5 => 6);
$n = array_unshift($arr, 10, "x");
print_r($arr);  // 输出：Array ( [0] => 10 [1] => x [f] => 1 [2] => 6 )
// 删除头部元素
$fruits = array("orange", "banana", "apple", "raspberry");
$fruit = array_shift($fruits);
print_r($fruit); // 输出：orange
print_r($fruits);  // 输出：Array ( [0] => banana [1] => apple [2] =>
```

```
    raspberry )
?>
```

3. array_push 函数

array_push 函数用于在参数数组$array 的尾部添加一个或多个元素，其语法格式如下：

```
int array_push(array &$array, mixed $value [, mixed $value]*)
```

新添加元素的键是数值键，从数组原有的最大数值键加 1 开始设置。如果原先没有数值键元素，则新添加元素的键从 0 开始设置。函数返回添加元素后数组的元素个数。

说明：若仅添加一个元素，更好的方法是采用$array[] = $value 的形式。

4. array_pop 函数

array_pop 函数用于从参数数组$array 的尾部删除一个元素，其语法格式如下：

```
mixed array_pop(array &$array)
```

函数返回被删除元素的值。若数组原先是空的，函数返回 NULL。

【例 10-7】 在数组尾部添加和删除元素。代码如下：

```
<?php
// 在尾部添加元素
$arr = array("f" => 1, 5 => 6);
$n = array_push($arr, 10, "x");
print_r($arr);   // 输出：Array ( [f] => 1 [5] => 6 [6] => 10 [7] => x )
// 删除尾部元素
$fruits = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($fruits);
print_r($fruit); // 输出：raspberry
print_r($fruits);// 输出：Array ( [0] => orange [1] => banana [2] => apple )
?>
```

10.4 遍历数组

遍历数组就是要访问数组中的每个元素。下面首先介绍数组指针移动和访问当前元素的若干函数，然后介绍使用 for、while 和 foreach 等语句遍历数组的方法。

10.4.1 数组指针

每个数组有一个内部指针，数组指针所指的元素称为该数组的当前元素。当创建一个数组时，数组指针初始指向第一个元素。使用下面函数可以移动数组指针的位置。

- mixed next(array &$array)

移动参数数组的内部指针至下一个元素，并返回下一个元素的值。

- mixed prev(array &$array)

移动参数数组的内部指针至上一个元素，并返回上一个元素的值。

- mixed reset(array &$array)

移动参数数组的内部指针至第一个元素，并返回第一个元素的值。

- mixed end(array &$array)

移动参数数组的内部指针至最后一个元素，并返回最后一个元素的值。

上面这些函数的共同特点是，首先移动数组指针，然后返回指针所指元素的值。如果指针移出了数组的范围（如指向了最后一个元素的后面位置、第一个元素的前面位置），或者数组为空（不含任何元素），那么函数将返回 false。

下面函数可以返回当前元素的键或值。

- mixed current(array &$array)

返回参数数组$array 当前元素的值。如果数组为空，或者数组指针所指位置超出了数组的范围，函数返回 false。

- mixed key(array &$array)

返回参数数组$array 当前元素的键。如果数组为空，或者数组指针所指位置超出了数组的范围，函数返回 NULL。

10.4.2 使用 for 语句遍历数组

如果数组各元素的键是从 0 开始的连续整数，即类似于一般计算机语言中的数组，那么可以使用 for 语句遍历之。

【例 10-8】 用 for 语句遍历连续整数键数组。代码如下：

```
<?php
$fruit = array('apple', 'banana', 'cranberry');
$c = count($fruit);
for($i=0; $i<$c; $i++) {
   echo $fruit[$i]."<br />";
}
?>
```

其中，count 函数返回参数数组$fruit 的元素个数。

10.4.3 使用 while 语句遍历数组

如果数组各元素的键不是整数，或者不是连续的整数，则无法使用 for 语句来遍历之。此时，可以使用 while 语句结合数组指针移动等函数来访问数组各元素。下面代码可以遍历数组$array：

```
reset($array);
while($current = current($array)) {
    $key = key($array);
    ...
    next($arr);
}
```

还有一种更有效率的方法是使用 each 函数并结合 list 语言结构。each 函数的语法格式如下：

```
array each(array &$array)
```

函数返回参数数组$array 的当前元素，然后移动数组指针至下一个元素。如果数组为空，或者数组指针所指位置超出了数组的范围，函数返回 false。

通常，each 函数返回的结果是一个包含 4 个元素的数组。这 4 个元素的内容如下：

键	值
1	<value>
value	<value>
0	<key>
key	<key>

其中，键为 0 和 key 的两个元素的值是数组$array 的当前元素的键，键为 1 和 value 的两个元素的值是数组$array 的当前元素的值。

就如同 array，list 并不是函数，而是一种语言结构，其一般语法格式如下：

```
list(mixed $var1 [, mixed $var2]*) = $array
```

其功能是用赋值号右边数组各元素的值给 list 中列出的一组变量赋值。一般来说，如果 list 列出了 *n* 个变量，那么赋值号右边的数组应该存在键为 0、1、…、*n*-1 的各元素。其赋值过程是从后往前的，最后一个变量最先被赋值，第一个变量最后被赋值，即数组元素$array[*n*-1]的值被最先赋给变量$varn，数组元素$array[0]的值被最后赋给变量$var1。如果不存在相应的数组元素，list 中对应变量的值为 NULL。

【例 10-9】 用 while 语句遍历数组。代码如下：

```
<?php
$fruit = array('a' => 'apple', 'b' => 'banana', 'c' => 'cranberry');
while($key=key($fruit)) {
   echo $key."=>".current($fruit)."<br />";
   next($fruit);
}
// 使用 each 函数和 list 语言结构
reset($fruit);
while(list($key, $value) = each($fruit)) {
   echo $key."=>".$value."<br />";
}
?>
```

分析一下上面代码中表达式 list($key, $value) = each($fruit)的计算。这里 list 列出两个变量：$key 和$value。如果 each 函数返回一个数组，则首先将其键为 1 的元素的值赋变量$value，然后将其键为 0 的元素的值赋给变量$key。整个表达式的值为 each 函数返回的数组。如果 each 函数返回 false，那么变量$key 或$value 的值均设置为 NULL，整个表达式的

值为 false。

10.4.4 使用 foreach 语句遍历数组

foreach 是一个专门用于遍历数组的循环语句，其语法格式如下：

格式 1：

```
foreach(<array_expression> as <$value>) <statement>
```

格式 2：

```
foreach(<array_expression> as <$key>=><$value>) <statement>
```

格式 1 遍历指定的数组$array_expression。每次执行循环体前，当前元素的值被赋给指定变量$value，然后数组指针移至下一个元素。格式 2 的功能与此类似，不同的是，除了会把当前元素的值赋给变量$value，还会将当前元素的键赋给变量$key。

foreach 语句在执行一开始，会首先将数组$array_expression 的内部指针移至第一个元素。因此不需要在 foreach 语句之前调用 reset 函数重置要遍历的数组。

【例 10-10】 使用 foreach 语句遍历数组。代码如下：

```
<?php
$fruit = array('a' => 'apple', 'b' => 'banana', 'c' => 'cranberry');
foreach ($fruit as $key=>$value) {
  echo $key."=>".$value."<br />";
}
?>
```

下面是代码运行的输出结果：

```
a=>apple
b=>banana
c=>cranberry
```

10.5 数组运算符

PHP 提供了一些数组运算符，可以实现数组的联合运算以及相等和不相等的比较，如表 10-1 所示。

表 10-1 数组运算符

运算符	名称	例子	结　果
+	联合	$a + $b	返回一个包含了$a 和$b 两个数组中所有元素的数组
= =	相等	$a = = $b	如果数组$a 和数组$b 具有相同的键-值对，返回 true
= = =	全等	$a = = = $b	如果数组$a 和数组$b 具有相同的键-值对以及相同的顺序和类型，返回 true

续表

运算符	名称	例子	结　　果
!= （<>）	不相等	$a != $b	如果数组$a 和数组$b 不相等，返回 true
!==	不全等	$a !== $b	如果数组$a 和数组$b 不全等，返回 true

对于联合运算（+），结果数组首先会包含左操作数（$a）的所有元素，然后再将右操作数（$b）中的各元素逐个添加到结果数组中。如果两个数组存在具有相同键的元素，那么取左操作数（$a）中的元素，而忽略右操作数（$b）中的相应元素。

两个数组的相等比较归结为两个数组中元素的相等比较。如果两个元素的键和值都相等，则两个元素相等。两个数组相等是指某数组中的一个元素在另一个数组中总是存在相等的元素。数组的相等比较不考虑元素在数组中的位置。

两个数组的全等比较归结为两个数组中元素的全等比较。如果两个元素的键和值都全等，则两个元素全等。两个数组全等是指两个数组的对应元素都全等。数组的全等比较需考虑元素在数组中的位置。

【例 10-11】 使用数组运算符。代码如下：

```
<?php
/* 联合运算符的使用 */
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");
$c = $a + $b;
print_r($c); // 输出: Array ( [a] => apple [b] => banana [c] => cherry )
$c = $b + $a;
print_r($c); // 输出: Array ( [a] => pear [b] => strawberry [c] => cherry )
/* 数组的相等（不相等）、全等（不全等）比较 */
$a = array("apple", "banana");
$b = array(1 => "banana", "0" => "apple");
var_dump($a == $b);     // 输出: bool(true)
var_dump($a === $b);    // 输出: bool(false)
$c = array("x" => 10, "y" => "20");
$d = array("x" => 10, "y" => 20);
var_dump($c != $d);     // 输出: bool(false)
var_dump($c !== $d);    // 输出: bool(true)
?>
```

10.6　数 组 排 序

数组排序是指对数组各元素进行排序。PHP 提供了许多函数，可以根据元素的值或键对数组进行升序或降序排序，也可以实现随机排序和反向排序。另外，还能够根据用户自定义的比较函数对数组进行特定的排序。

10.6.1 sort 函数

sort 函数的语法格式如下：

```
bool sort(array &$array [, int $sort_flags])
```

函数根据元素值对参数数组$array 的各元素从小到大进行排序。排序后，各元素的键重新设置为从 0 开始的连续整数。若排序成功，函数返回 true；否则函数返回 false。

可选参数$sort_flags 指定排序时的数据比较特性，其可能的取值如下。

- SORT_REGULAR：默认值。正常比较（不改变类型）。
- SORT_NUMERIC：数值化比较。
- SORT_STRING：字符串化比较。
- SORT_LOCALE_STRING：基于当前场所的字符串化比较。
- SORT_NATURAL：自然顺序比较。
- SORT_FLAG_CASE：能与 SORT_STRING 或 SORT_NATURAL 组合，以便比较时不区分字母大小写。

注意： *在排序混合类型的数组时，sort 函数可能会产生不可预期的结果。*

【例 10-12】 使用 sort 函数排序数组。代码如下：

```
<?php
$a = array(11, 19, 15, 30, -1, 28);
sort($a);
echo "<br />正常比较顺序：<br />";
print_r($a);
$b = $c = array(11, -1, "ch1", "ch2", "ch10", "12");
sort($b, SORT_NUMERIC);
echo "<br />数值化比较顺序：<br />";
print_r($b);
sort($c, SORT_STRING);
echo "<br />字符串比较顺序：<br />";
print_r($c);
$d = array("item1", "item10", "item3", "item20", "item2");
sort($d, SORT_NATURAL);
echo "<br />自然顺序：<br />";
print_r($d);
?>
```

下面是代码运行的输出结果：

```
正常比较顺序：
Array ( [0] => -1 [1] => 11 [2] => 15 [3] => 19 [4] => 28 [5] => 30 )
数值化比较顺序：
Array ( [0] => -1 [1] => ch10 [2] => ch2 [3] => ch1 [4] => 11 [5] => 12 )
字符串比较顺序：
```

```
Array ( [0] => -1 [1] => 11 [2] => 12 [3] => ch1 [4] => ch10 [5] => ch2 )
自然顺序：
Array ( [0] => item1 [1] => item2 [2] => item3 [3] => item10 [4] => item20 )
```

10.6.2 asort 和 ksort 函数

asort 函数的语法格式如下：

```
bool asort(array &$array [, int $sort_flags])
```

该函数根据元素值对参数数组$array 的各元素从小到大进行排序。排序时，各元素的键和值保持关联。若排序成功，函数返回 true；否则函数返回 false。

ksort 函数的语法格式如下：

```
bool ksort(array &$array [, int $sort_flags])
```

该函数根据元素键对参数数组$array 的各元素从小到大进行排序。排序时，各元素的值和键保持关联。若排序成功，函数返回 true；否则函数返回 false。

如同 sort 函数，asort 和 ksort 函数也可以通过设置可选参数$sort_flags 来改变排序的行为。

【例 10-13】 使用 asort 函数和 ksort 函数。代码如下：

```
<?php
$a = $b = array("apple" => 12.5, "orange" => 9.8, "banana" => 15.2);
asort($a);
print_r($a); // 输出：Array ( [orange] => 9.8 [apple] => 12.5 [banana] => 
   15.2 )
ksort($b);
print_r($b); // 输出：Array ( [apple] => 12.5 [banana] => 15.2 [orange] => 
   9.8 )
?>
```

10.6.3 降序排序

前面介绍的 sort、asort 和 ksort 函数有一个共同的特点，即都是对参数数组各元素进行从小到大的升序排序。与此相应，rsort、arsort 和 krsort 函数可以对参数数组各元素进行从大到小的降序排序。它们的语法格式如下：

```
bool rsort(array &$array [, int $sort_flags])
bool arsort(array &$array [, int $sort_flags])
bool krsort(array &$array [, int $sort_flags])
```

函数 rsort 按元素值对参数数组$array 的各元素进行降序排序。排序后，各元素的键重新设置为从 0 开始的连续整数。

函数 arsort 按元素值对参数数组$array 的各元素进行降序排序。排序时，各元素的键和

值保持关联。

函数 krsort 按元素键对参数数组$array 的各元素进行降序排序。排序时，各元素的值和键保持关联。

10.6.4 随机排序和反向排序

这里介绍其他两种排序：随机排序和反向排序。

1. 随机排序

随机排序由 shuffle 函数实现，该函数对参数数组$array 的各元素进行随机排序。格式如下：

```
bool shuffle(array &$array)
```

排序后，各元素的键重新设置为从 0 开始的连续整数。若排序成功，函数返回 true；否则函数返回 false。

2. 反向排序

反向排序由 array_reverse 函数实现，该函数返回一个内容与参数数组相同但顺序相反的数组。格式如下：

```
array array_reverse(array $array [, bool $preserve_keys])
```

可选参数$preserve_keys 的默认值是 false。在这种情况下，各元素的原来的整数键被丢弃，而被重新设置为从 0 开始的连续整数。各元素的字符串键被保留。

如果将 preserve_keys 设置为 true，则整数键和字符串键都将被保留。

10.6.5 用户自定义排序

在实际应用中，系统预定的排序方式不一定能满足需求。有时也可能有其他的排序需求，比如要对各字符串根据其长度进行排序。这时需要用户自定义排序。

要实现自定义排序，首先需要编写比较函数。比较函数接收两个参数值、实现对两个值的比较，并返回大于 0、等于 0 或小于 0 的整数，分别表示第 1 个参数值大于、等于和小于第 2 个参数值。

然后利用相应的排序函数完成数组的自定义排序。能完成自定义排序的函数包括 usort、uasort 和 uksort，这 3 个函数的语法格式如下：

```
bool usort(array &$array, callable $value_compare_func)
bool uasort(array &$array, callable $value_compare_func)
bool uksort(array &$array, callable $key_compare_func)
```

这 3 个函数与之前介绍的 sort、asort 和 ksort 函数分别相对应，功能都相似，只是它们都需要指定一个比较函数，并根据该函数来比较两个数组元素的大小。

【例 10-14】 根据字符串长度对数组中各字符串元素进行排序。代码如下：

```
1. <?php
2. // 比较函数：根据长度决定两个字符串的大小
```

```
function compare($x, $y) {
  if(strlen($x) == strlen($y)) {
     return 0;
  } else if(strlen($x) < strlen($y)) {
     return -1;
  } else {
     return 1;
   }
}
// 利用 usort 函数排序数组
$a = $b = array("computer", "one", "city", "people");
usort($a, "compare");
print_r($a);    // 输出：Array ( [0] => one [1] => city [2] => people [3] => computer )
uasort($b, "compare");
print_r($b);    // 输出：Array ( [1] => one [2] => city [3] => people [0] => computer )
?>
```

这里，自定义排序函数的参数有两个：一是需要排序的数组$a 或$b，二是比较函数的名称"compare"。在排序时，PHP 调用比较函数完成对两个元素的比较。

有时候也需要对多维数组进行排序。在多维数组中，每个元素都可能是一个数组。这时也可以借助于用户自定义排序，即先定义一个比较函数，可以比较两个内部数组的大小，然后再利用相应的自定义排序函数实现对多维数组的排序。

10.7　并集、交集和差集

数组是数据或元素的集合。本节介绍针对数组的集合运算，即利用 PHP 内置函数求数组的并集、交集和差集。

10.7.1　求数组的并集

合并数组可以将若干数组合并在一起，返回一个包含各数组所有元素的数组。合并时，各参数数组依次将其各元素按其原先的顺序追加到结果数组中，各数值键被重新设置为从 0 开始的连续整数。

实现合并数组功能的函数包括 array_merge 和 array_merge_recursive。

1. array_merge 函数

array_merge 函数的语法格式如下：

```
array array_merge(array $array1 [, array $array2]*)
```

该函数在合并追加一个元素时，如果是数值键，那么追加新元素并重新设置键；如果是字符串键且之前没有具有相同键的元素，那么追加新元素；如果是字符串键但之前已有

相同键的元素，那么用新的元素值替换原有的值。

比较数组联合运算（+）和数组合并操作：

（1）对于联合运算，无论是数值键还是字符串键，如果出现两个元素的键相同，那么只保留左操作数数组的元素的值。

（2）对于合并操作，数值键的元素总是作为新元素放入结果数组中，且键被重新设置。字符串键的元素则会做替换操作。

2. array_merge_recursive 函数

array_merge_recursive 函数的语法格式如下：

```
array array_merge_recursive(array $array1 [, array $array2]*)
```

该函数与 array_merge 函数的功能大致相同，区别在于：如果合并的数组中存在有相同字符串键的元素，不是执行替换操作，而是做合并操作，即将两个值合并成一个数组作为元素值。如果两个值本身就是数组，就将两个数组合并成一个数组作为元素值。所以这是一种递归合并。

【例 10-15】求数组的并集。使用 array_merge 和 array_merge_recursive 函数。代码如下：

```
<?php
$ar1 = array("shape" => "circle", "color" => array("favorite" => "red"), 5);
$ar2 = array(10, "color" => array("favorite" => "green", "blue"), "shape" => "rectangle");
$result1 = array_merge($ar1, $ar2);
print_r($result1);
/* 输出:
Array (
  [shape] => rectangle
  [color] => Array ( [favorite] => green [0] => blue )
  [0] => 5
  [1] => 10
)
*/
$result2 = array_merge_recursive($ar1, $ar2);
print_r($result2);
/* 输出:
Array (
  [shape] => Array ( [0] => circle [1] => rectangle )
  [color] => Array ( [favorite] => Array ( [0] => red [1] => green ) [0] => blue )
  [0] => 5
  [1] => 10
)
*/
?>
```

10.7.2　求数组的交集

数组的交集是指在各数组中都存在的元素的集合。PHP 提供求数组交集的函数，包括 array_intersect 和 array_intersect_asso。

1. array_intersect 函数

array_intersect 函数的语法格式如下：

```
array array_intersect(array $array1, array $array2 [, array $array3]*)
```

该函数返回一个数组，这个结果数组仅包含第一个参数数组中其值在其他所有参数数组中都出现的元素。

2. array_intersect_assoc 函数

array_intersect_assoc 函数的语法格式如下：

```
array array_intersect_assoc(array $array1, array $array2 [, array $array3]*)
```

该函数返回一个数组，这个结果数组仅包含第一个参数数组中其键-值对在其他所有参数数组中都出现的元素。也就是说，在比较时不仅要考虑值还要考虑键。

【例 10-16】 求数组的交集。使用 array_intersect 和 array_intersect_asso 函数。代码如下：

```
<?php
$arr1 = array("a" => "green", "blue", "red");
$arr2 = array("b" => "green", "red", "yellow");
$result1 = array_intersect($arr1, $arr2);
print_r($result1);   // 输出: Array ( [a] => green [1] => red )

$arr3 = array("a" => "green", "blue", "x" => "red");
$arr4 = array("b" => "green", "x" => "red", "yellow" , "blue");
$result2 = array_intersect_assoc($arr3, $arr4);
print_r($result2);  // 输出: Array ( [x] => red )
?>
```

注意：结果数组$result2 不包含 blue，因为在第 1 个参数数组$arr3 中相应元素的键为 0，在第 2 个参数数组$arr4 中相应元素的键为 1。

10.7.3　求数组的差集

数组的差集是指在第 1 个数组中出现而在其他数组中不存在的元素的集合。求数组的差集的函数包括：array_diff 和 array_diff_assoc。

1. array_diff 函数

array_diff 函数的语法格式如下：

```
array array_diff(array $array1, array $array2 [, array $array3]*)
```

该函数返回一个数组，这个结果数组仅包含第一个参数数组中其值没有在任何其他参数数组中出现的元素。

2. array_diff_assoc 函数

array_diff_assoc 函数的语法格式如下：

```
array array_diff_assoc(array $array1, array $array2 [, array $array3]*)
```

该函数返回一个数组，这个结果数组仅包含第一个参数数组中其键-值对没有在任何其他参数数组中出现的元素。也就是说，在比较时不仅要考虑值还要考虑键。

【例 10-17】 求数组的差集。使用 array_diff 和 array_diff_assoc 函数。代码如下：

```
<?php
$arr1 = array("OH", "CA", "NY", "HI", "CT");
$arr2 = array("OH", "HI", "CA", "NE", "IA");
$arr3 = array("TX", "MD", "NY", "OH", "HI");
$result1 = array_diff($arr1, $arr2, $arr3);
print_r($result1);   // 输出: Array ( [4] => CT )
$result2 = array_diff_assoc($arr1, $arr2, $arr3);
print_r($result2);   // 输出: Array ( [1] => CA [3] => HI [4] => CT )
?>
```

10.8 其他常用的数组函数

本节介绍其他一些常用的数组函数，包括计数与统计、结合与拆分、变量与数组元素的转换以及用自定义函数处理数组各元素等内容。

10.8.1 计数与统计

这里介绍 count、array_count_values 和 array_sum 这 3 个 PHP 内置函数。

1. count 函数

count 函数返回数组中元素的个数，其语法格式如下：

```
int count(array $array [, int $mode])
```

可选参数$mode 的取值有下面两种。

- COUNT_NORMAL：默认值。仅统计最外层数组的元素个数。
- COUNT_RECURSIVE：递归统计各层数组的元素个数。

【例 10-18】 使用 count 函数。代码如下：

```
<?php
$food = array('fruits' => array('orange', 'banana', 'apple'),
 'veggie' => array('carrot', 'collard', 'pea'));
echo count($food);                  // 输出: 2
echo count($food, COUNT_RECURSIVE); // 输出: 8
?>
```

第一个是正常计数，返回外层数组的元素个数，即 2 个内部数组。第二个是递归计数，

首先外层数组的元素个数为 2，而两个内部数组的元素个数都为 3，总数为 8。

注意：sizeof 函数是 count 函数的别名，具有相同的功能。

2. array_count_values 函数

array_count_values 函数的语法格式如下：

```
array array_count_values(array $array)
```

该函数返回一个数组，其元素的键是参数数组中元素的值，相应的值是该键作为元素值在参数数组中出现的次数（频度）。

【例 10-19】 使用 array_count_values 函数。代码如下：

```
<?php
$a = array(1, "hello", 1, "world", "hello");
$b = array_count_values($a);
print_r($b); // 输出：Array ( [1] => 2 [hello] => 2 [world] => 1 )
?>
```

3. array_sum 函数

array_sum 函数的语法格式如下：

```
number array_sum(array $array)
```

该函数计算参数数组中各元素值的和，返回一个整数或浮点数。

对非数值型元素值，函数会将其转换成数值（大多数情况为 0）。函数对内部数组的元素值不会进行递归累加。

【例 10-20】 使用 array_sum 函数。代码如下：

```
<?php
$a = array(5, "hello", 3, "12", array(2, 8), 10);
$sum = array_sum($a);
print_r($sum);   // 输出：30
?>
```

其中，字符串"12"会被转换成数值进行累加，内部数组中的 2 和 8 不会进行累加。

10.8.2 结合与拆分

这里介绍有关数组结合和拆分的几个函数，包括 array_combine 函数、array_slice 函数、array_splice 函数和 array_chunk 函数。

1. array_combine 函数

array_combine 函数接收两个大小相等的数组，经过结合返回一个新数组。结果数组中元素的键是第一个参数数组中元素的值，元素的值是第二个参数数组中对应位置上元素的值。其语法格式如下：

```
array array_combine(array $keys, array $values)
```

如果两个参数数组的大小不同，函数产生 Warning 信息，并返回 false。如果第一个参数数组中的元素值不能作为合法的键（如浮点数），那么会被自动转换成字符串。

【例 10-21】 使用 array_combine 函数。代码如下：

```
<?php
$a = array('green', 'red', 'yellow');
$b = array('avocado', 'apple', 'banana');
$c = array_combine($a, $b);
print_r($c); // 输出：Array ( [green] => avocado [red] => apple [yellow] => banana )
?>
```

2．array_slice 函数

array_slice 函数从参数数组$array 中提取连续的若干元素，组成一个新数组返回。其语法格式如下：

```
array array_slice(array $array, int $offset [, int $length [, bool $preserve_keys]])
```

参数$offset 和$length 分别指定要提取的起始元素位置和元素个数。

若$offset 是非负整数，则起始元素是参数数组的第$offset+1 个元素；若$offset 是负整数，则起始元素是参数数组倒数第–$offset 个元素。

若$length 是非负整数，则顺序取$length 个元素，或者取至数组尾（如果没有更多的元素）。若$length 是负整数，则顺序取若干元素，但保留数组末尾的–$length 个元素。若缺省$length（其默认值为 NULL），则取至数组尾。

参数$preserve_keys 是可选的。若缺省该参数（其默认值为 false），那么结果数组中各数值键元素的键被重新设置为从 0 开始的连续整数。若想保留这些元素原有的数值键，可将该参数设置为 true。

【例 10-22】 使用 array_slice 函数。代码如下：

```
<?php
$input = array("a" => 10, 20, "b" => 30, 40, "c" => 50, 60, "d" => 70);
$output1 = array_slice($input, 2, 4);
print_r($output1);// 输出：Array ( [b] => 30 [0] => 40 [c] => 50 [1] => 60 )
$output2 = array_slice($input, -4, NULL, true);
print_r($output2);   // 输出：Array ( [1] => 40 [c] => 50 [2] => 60 [d] => 70 )
$output3 = array_slice($input, 2, -3);
print_r($output3);   // 输出：Array ( [b] => 30 [0] => 40 )
?>
```

3．array_splice 函数

array_splice 函数可以删除数组中连续的若干元素，并可用另一个数组的元素替换这些被删除的元素。其语法格式如下：

```
array array_splice(array &$input, int $offset [, int $length [, mixed $replacement]])
```

其中，参数$input、$offset 和$length 的含义与 array_slice 函数中的相同。与 array_slice 函数相比，该函数有如下不同：

（1）不只是提取指定元素，而是删除这些函数，但删除的元素仍会存放在一个数组中返回。

（2）该函数没有可选参数$preserve_keys。返回的数组中，各数值键元素不能保留原先的键，而总是被重新设置为从 0 开始的连续整数。

（3）如果指定数组$replacement，那么该数组中的元素将被插入至被删除元素的位置，这些被插入元素原先的键不被保留，而变成数值键。$replacement 也可以仅是一个值。

参数数组$input 经过删除部分元素及插入一些元素后，其各数值键元素的键会被重新设置为从 0 开始的连续整数。

【例 10-23】 使用 array_splice 函数。代码如下：

```
<?php
$replacement = array("x" => 100, "y" => 200);
$input = array("a" => 10, 20, "b" => 30, 40, "c" => 50, 60, "d" => 70);
array_splice($input, 2, 3, $replacement);
print_r($input);
//输出：Array ( [a] => 10 [0] => 20 [1] => 100 [2] => 200 [3] => 60 [d] => 70)
$input = array("a" => 10, 20, "b" => 30, 40, "c" => 50, 60, "d" => 70);
array_splice($input, 2, -3, 100);
print_r($input);
// 输出：Array ([a] => 10 [0] => 20 [1] => 100 [c] => 50 [2] => 60 [d] => 70)
?>
```

4. array_chunk 函数

array_chunk 函数可以将参数数组$array 划分成若干大小为$size 的数组（最后一个数组的大小可能小于$size），这些部分数组作为元素被保存在一个新的数组返回。其语法格式如下：

```
array array_chunk(array $array, int $size [, bool $preserve_keys])
```

参数$preserve_keys 是可选的。若忽略该参数（默认值为 false），每个部分数组中各元素的键被重新设置为 0 开始的连续的整数键。如果将$preserve_keys 设置为 true，所有元素保留原先的键。

【例 10-24】 使用 array_chunk 函数。代码如下：

```
<?php
$input = array("a" => 10, 20, "b" => 30, 40, "c" => 50, 60, "d" => 70);
$result1 = array_chunk($input, 3);
print_r($result1);
/* 输出：
```

```
Array (
   [0] => Array ( [0] => 10 [1] => 20 [2] => 30 )
   [1] => Array ( [0] => 40 [1] => 50 [2] => 60 )
   [2] => Array ( [0] => 70 )
)
*/
$result2 = array_chunk($input, 3, true);
print_r($result2);
/* 输出:
Array (
   [0] => Array ( [a] => 10 [0] => 20 [b] => 30 )
   [1] => Array ( [1] => 40 [c] => 50 [2] => 60 )
   [2] => Array ( [d] => 70 )
)
*/
?>
```

10.8.3 变量与数组元素的转换

compact 函数可以将一些变量组合成一个数组，extract 函数可以基于数组元素产生变量。

1. compact 函数

compact 函数可以基于一个或多个变量创建一个数组返回。结果数组中，每个元素的键是变量名，值是变量值。其语法格式如下：

```
array compact(mixed $varname1 [, mixed $varname2]*)
```

函数的每一个参数可以是表示变量名的字符串，函数会为每个变量在结果数组中添加一个相应的元素。如果不存在这样的变量，则直接跳过。

函数中的某个参数也可以是数组，这时函数将递归处理该数组中的元素，即数组中的元素值也应该是某个变量的名称，函数将基于该变量名在结果数组中添加一个相应的元素。

【例 10-25】 使用 compact 函数。代码如下：

```
<?php
$city = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";
$location_vars = array("city", "state");
$result = compact("event", "nothing_here", $location_vars);
print_r($result);
// 输出: Array ([event] => SIGGRAPH [city] => San Francisco [state] => CA )
?>
```

2. extract 函数

extract 函数根据参数数组$array 的元素创建变量，元素的键作为变量名，元素的值作为变量值。其语法格式如下：

```
int extract(array &$array [, int $flags [, string $prefix]])
```

这里，元素的键可能是一个非法的变量名（如数值键、包含空格的字符串等），也可能原先已有同名的变量（冲突）。可选参数$flags 和$prefix 的设置将决定如何处理这些情况。除非将 flags 指定为 EXTR_PREFIX_ALL 或 EXTR_PREFIX_INVALID，其他情况函数将忽略非法变量名的元素。函数返回成功创建的变量数。

参数 flags 的可能取值有下面几种。

- EXTR_OVERWRITE：默认值。若出现变量名冲突，覆盖原有变量。
- EXTR_SKIP：若出现变量名冲突，保留原有变量。
- EXTR_PREFIX_SAME：若出现变量名冲突，新建变量名前添加指定前缀。
- EXTR_IF_EXISTS：仅当出现变量名冲突时，覆盖原有变量；其他情况下跳过。
- EXTR_PREFIX_IF_EXISTS：仅当出现变量名冲突时，创建新变量，变量名前添加指定前缀。
- EXTR_PREFIX_ALL：考虑所有元素，所有新建变量名添加指定前缀。
- EXTR_PREFIX_INVALID：考虑所有元素，但仅在非法变量名前添加指定前缀。
- EXTR_REFS：以引用方式提取变量，即数组元素与新产生的变量引用相同的内存单元。该值可以独立使用或与上面任何值配合使用。

参数$prefix 指定新建变量名的前缀，前缀和键名之间用下画线连接。如果添加前缀后的变量名仍然是非法的，跳过该元素。

【例 10-26】 使用 extract 函数。代码如下：

```
<?php
$input = array("a" => 10, 20, "b" => 30, 40, "c" => 50, 60, "d" => 70);
$n = extract($input);
echo "新建 $n 个变量：$a, $b, $c, $d";
// 输出：新建 4 个变量：10, 30, 50, 70
$n = extract($input, EXTR_PREFIX_INVALID, "x");
echo "<br />新建 $n 个变量：$a, $x_0, $b, $x_1, $c, $x_2, $d";
// 输出：新建 7 个变量：10, 20, 30, 40, 50, 60, 70
?>
```

10.8.4 用自定义函数处理数组各元素

array_walk 函数可以调用一个用户自定义函数来依次处理某个数组中的各元素。其语法格式如下：

```
bool array_walk(array &$array, callable $callback [, mixed $userdata])
```

参数$array 指定要被处理的数组，参数$callback 指定用户自定义函数。用户自定义函数至少定义两个形参，第 1 个形参接收当前被处理的数组元素的值，第 2 个形参接收当前被处理的数组元素的键。如果指定可选参数$userdata，用户自定义函数可以定义第 3 个形参，用于接收该参数值。

一般来说，用户自定义函数仅能修改数组元素的值，而不应该试图更改数组元素的键，也不应该更改数组的结构，如添加元素，重新排序元素等。

【例 10-27】 使用 array_walk 函数。代码如下：

```
<?php
// 用于处理数组元素的自定义函数。由于要修改元素的值，第 1 个参数采用引用传递
function test_alter(&$item, $key, $prefix) {
  $item = "$prefix: $item";
}

$fruits = array("a" => "orange", "b" => "banana", "c" => "apple");
array_walk($fruits, 'test_alter', 'fruit');
print_r($fruits);    // 输出: Array ( [a] => fruit: orange [b] => fruit: banana [c] => fruit: apple )
?>
```

10.9 实例：维护开课信息

本实例实现管理员子系统导航栏中“维护开课信息”菜单项的功能。当单击导航栏中“维护开课信息”菜单项时，将请求“维护开课信息”页面文件 opencourse_p.php。该页面包含两个视图，页面文件通过请求参数 v 的值决定呈现哪个视图。

第 1 个视图（v=1）是“添加开课信息”，对应模块文件 opencourse_input.php，可以呈现添加开课信息的表单、提供为管理员所属部门添加指定学期的开课信息的功能，其呈现效果如图 10-1 所示页面的主区部分。



图 10-1 “维护开课信息”页面的“添加开课信息”视图

第 2 个视图（v=2）是“开课列表”，对应模块文件 opencourse_list.php，可以呈现指定学期管理员所属部门负责的开课信息列表，并包含删除开课信息和更改开课状态的功能，其呈现效果如图 10-2 所示页面的主区部分。

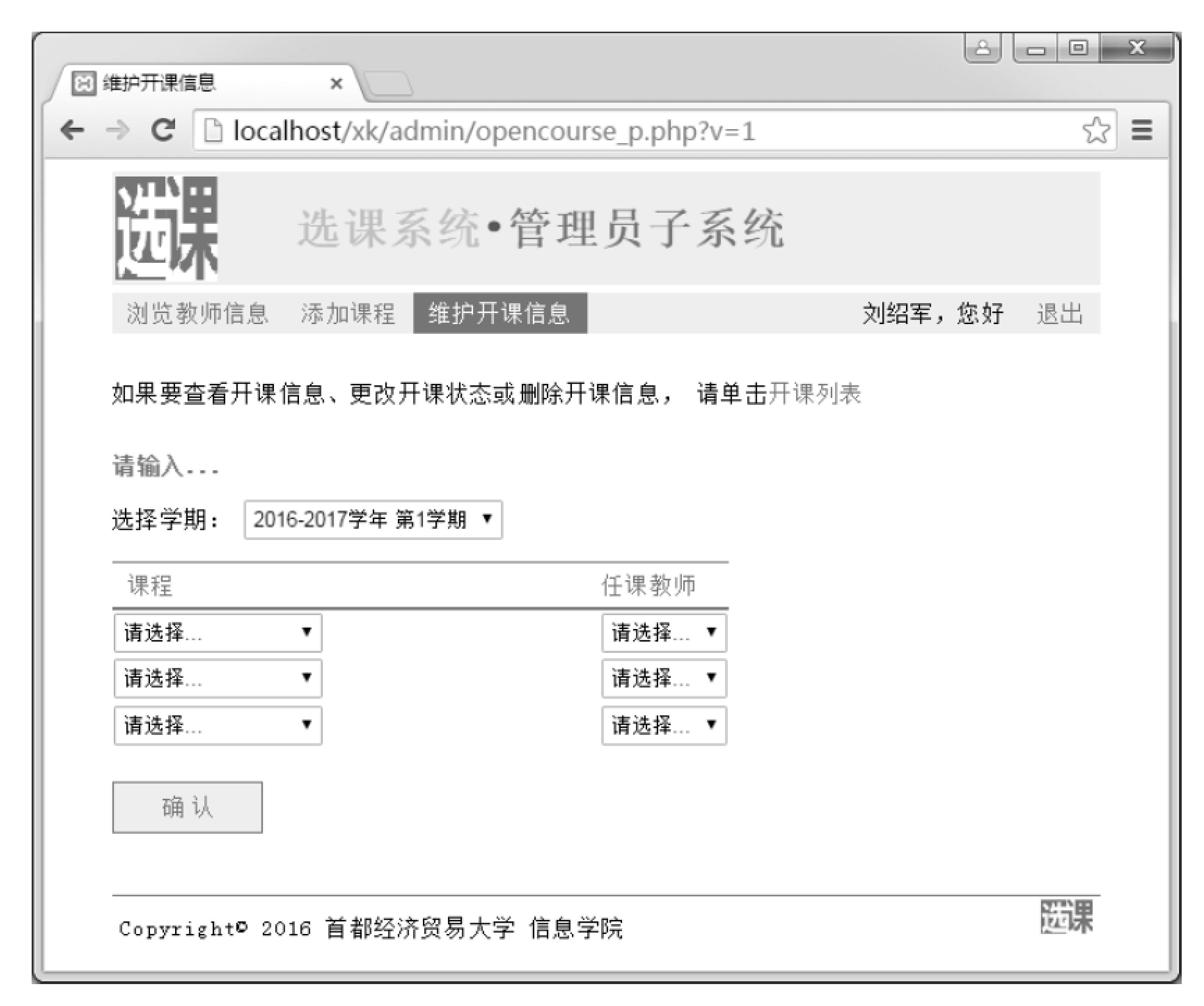


图 10-2 “维护开课信息”页面的“开课列表”视图

首先给出“维护开课信息”页面文件 opencourse_p.php 的代码。该文件利用 PHP 包含文件技术，通过包含页面头模块文件、导航栏模块文件、“添加开课信息”或“开课列表”模块文件以及页面脚模块文件组成完整的页面。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>维护开课信息</title>
    <link rel="stylesheet" type="text/css" href="/xk/css/xk.css"/>
  </head>
  <body>
    <?php
      session_start();
      $lb = @$_SESSION['lb'];
      if(empty($lb) || $lb !== '管理员') {
        header("Location: index_admin.php");
        exit();
      }
      $user = @$_SESSION['user'];
```

```
            $name = @$_SESSION['name'];
            $dept = @$_SESSION['dept'];
        ?>
        <?php include("header_admin.php"); ?>
        <?php
            $choice = 3;
            include("navigation_admin.php");
        ?>
        <div style="width: 90%; min-height: 150px; margin: 0 auto; padding: 5px;">
            <?php
                $v = @$_GET['v'];
                if(empty($v)) $v = "1";
                if($v === "1") {
                    include("opencourse_input.php");
                } else {
                    include("opencourse_list.php");
                }
            ?>
        </div>
        <?php include("footer_admin.php"); ?>
    </body>
</html>
```

然后给出“添加开课信息”模块文件 opencourse_input.php 的代码。该模块文件呈现于页面主区，是“维护开课信息”页面的第 1 个视图，当请求参数 v 等于 1 时被调用。

这里，表单中的“选择学期”选择列表的选项是基于当前年份的前后 3 个学期；“课程”选择列表的选项只包含管理员所属部门负责的课程；“任课教师”选择列表的选项只包含管理员所属部门的教师。

表单一次可以输入的开课数量可以在第 14 行代码处设置，这里为 3。当表单提交时，第 17 行至第 29 行代码会对提交的表单数据进行处理，对于课程和任课教师都已指定的开课信息，会试图添加到数据库。当处理完成后，会重新呈现该视图，并显示有关处理结果的提示信息。代码如下：

```
1. <!--
2. *  功能：管理员向数据库添加所属部门的开课信息
3. *  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/css
          /xk.css"/>
4. *       $dept：登录管理员的所属部门，如"信息学院"
5. -->
6. <?php
7. include_once '../lib/mysql.php';
8. /* 连接数据库 */
9. $mysqli = connect();
```

```
if ($mysqli->connect_errno) {
   echo "不能连接到数据库<br/>";
   exit();
}
$n = 3;                   // 设置一次最多可输入的开课数量
$title = "请输入...";     // 初始化提示信息
/* 处理表单数据，向数据库添加开课信息 */
if($_SERVER["REQUEST_METHOD"]==="POST" && array_key_exists("submit", $_POST)) {
   $count = 0;
   $term = $_POST["term"];
   for($i=0; $i<$n; $i++) {
      $cn = $_POST["cn$i"];
      $tn = $_POST["tn$i"];
      if($cn!=="" && $tn!=="") {// 课程和教师都已指定
         $sql = "INSERT INTO opencourse(term, cn, tn) VALUES('$term', '$cn', '$tn')";
         if(executeSql($mysqli, $sql)) $count++;
      }
   }
   $title = "已添加{$count}条开课信息，请继续...";
}
/* 根据当前年份，构建表单中“选择学期”选择列表中各选项的 HTML 代码 */
$year = getdate()['year'];
$value1 = ($year-1)."-".$year."-2"; $label1 = ($year-1)."-".$year."学年 第 2 学期";
$value2 = $year."-".($year+1)."-1"; $label2 = $year."-".($year+1)."学年 第 1 学期";
$value3 = $year."-".($year+1)."-2"; $label3 = $year."-".($year+1)."学年 第 2 学期";
if(empty($term)) $term = $value2;
$termoptions = "<option value='$value1'"
      .($term === $value1 ? " selected = 'selected'>": ">")."$label1</option>\r\n";
$termoptions .= "<option value='$value2'"
      .($term === $value2 ? " selected='selected'>" : ">")."$label2</option>\r\n";
$termoptions .= "<option value='$value3'"
      .($term === $value3 ? " selected='selected'>" : ">")."$label3</option>\r\n";
/* 根据当前管理员所属部门，构建表单中“课程”选择列表中各选项的 HTML 代码 */
$sql = "SELECT cn, cname FROM course c, teacher t "
     . "WHERE c.tn=t.tn and t.dept = '$dept' ORDER BY cn";
$result = executeSql($mysqli, $sql);
$cnoptions = "<option value=''>请选择...</option>";
```

```
while($row = $result->fetch_array()) {
   $cnoptions .= "<option value='".$row['cn']."'>".$row ['cname']."
   </option>\r\n";
}
/* 根据当前管理员所属部门，构建表单中“任课教师”选择列表中各选项的 HTML 代码 */
$sql = "SELECT tn, tname FROM teacher WHERE dept = '$dept' ORDER BY tn";
$result = executeSql($mysqli, $sql);
$tnoptions = "<option value=''>请选择...</option>\r\n";
while($row = $result->fetch_array()) {
   $tnoptions .= "<option value='".$row['tn']."'>".$row['tname'].
   "</option>\r\n";
}
$mysqli->close();
?>
<!-- 呈现开课信息录入表单 -->
<style type="text/css">
   table .c1 {width: 300px; text-align: left; padding-left: 10px}
   table .c2 {width: 80px; text-align: left}
</style>
<div style="margin: 20px 0 30px 0">
   如果要查看开课信息、更改开课状态或删除开课信息，
   请单击<a href="opencourse_p.php?v=2">开课列表</a>
</div>
<form method="POST">
   <div class="title"><?php echo $title ?></div>
   <p>
   <label for="i1">选择学期：</label>
   <select id="i1" name="term"><?php echo $termoptions; ?></select>
   </p>
   <table>
   <thead>
     <tr><td class="c1">课程</td><td class="c2">任课教师</td></tr>
   </thead>
   <tbody>
     <?php for($i=0; $i<$n; $i++) { ?>
     <tr>
     <td><select name='<?php echo "cn$i" ?>'><?php echo $cnoptions ?>
    </select></td>
     <td><select name='<?php echo "tn$i" ?>'><?php echo $tnoptions ?>
    </select></td>
     </tr>
     <?php } ?>
   </tbody>
   </table>
   <p style="margin-top: 20px">
```

```
        <input type="submit" class="big" name="submit" value="确 认"/>
    </p>
</form>
```

最后给出“开课列表”模块文件 opencourse_list.php 的代码。该模块文件呈现于页面主区，是“维护开课信息”页面的第 2 个视图，当请求参数 v 等于 2（不等于 1）时被调用。

该视图呈现指定学期管理员所属部门负责的开课信息，并包含删除开课信息和更改开课状态的功能。只有处于“选课”和“教学”状态的开课信息才能被删除。当单击“删除”按钮时，将删除该按钮所在行的开课信息，同时删除学生选课表（elective）中相关的选课信息。更改开课状态是指在呈现的开课列表中，将所有处于“选课”状态的开课课程更改为“教学”状态。代码如下：

```
<!--
*  功能：呈现登录管理员所属部门的开课信息列表，
*         并包含删除开课信息和更改开课状态的功能
*  输入：链入外部样式表<link rel="stylesheet" type="text/css" href="/xk/css
          /xk.css"/>
*         $dept：登录管理员的所属部门，如"信息学院"
-->
<?php
include_once '../lib/mysql.php';
@ session_start();
/* 连接数据库 */
$mysqli = connect();
if ($mysqli->connect_errno) {
   echo "不能连接到数据库<br/>";
   exit();
}
/* 处理各个 POST 请求 */
if ($_SERVER["REQUEST_METHOD"]==="POST") {
   if(array_key_exists("submit1", $_POST)) {    // 处理提交的学期
      $term = $_POST['term'];
      $_SESSION['term'] = $term;
      $_SESSION['ocs'] = getOpenCourseData($mysqli, $dept, $term);
   } elseif(array_key_exists("lid", $_POST)) { // 处理“删除”请求
      $lid = $_POST['lid'];
      $mysqli->begin_transaction();
      $sql = "DELETE FROM elective WHERE lid='$lid'";
      executeSql($mysqli, $sql);
      $sql = "DELETE FROM opencourse WHERE lid='$lid'";
      executeSql($mysqli, $sql);
      $mysqli->commit();
      $term = $_SESSION['term'];
      $_SESSION['ocs'] = getOpenCourseData($mysqli, $dept, $term);
   } elseif(array_key_exists("cs", $_POST)) {  // 处理状态更改请求
```

```
        $term = $_SESSION['term'];
        $sql = "UPDATE opencourse SET status='2' WHERE term='$term' AND status='1' AND "
                . "tn IN (SELECT tn FROM teacher WHERE dept='$dept')";
        executeSql($mysqli, $sql);
        $_SESSION['ocs'] = getOpenCourseData($mysqli, $dept, $term);
    }
}
/* 处理 GET 请求：当从其他页面或视图回到该视图时 */
if ($_SERVER["REQUEST_METHOD"] == "GET") {
    if(array_key_exists("term", $_SESSION)) {
        $term = $_SESSION['term'];
        $_SESSION['ocs'] = getOpenCourseData($mysqli, $dept, $term);
    }
}
$mysqli->close();
// 根据当前年份，构建表单中“选择学期”选择列表中各选项的 HTML 代码
$year = getdate()['year'];
$value1 = ($year-1)."-".$year."-2"; $label1 = ($year-1)."-".$year."学年 第2学期";
$value2 = $year."-".($year+1)."-1"; $label2 = $year."-".($year+1)."学年 第1学期";
$value3 = $year."-".($year+1)."-2"; $label3 = $year."-".($year+1)."学年 第2学期";
@$term = $_SESSION['term'];
if(empty($term)) $term = $value2;
$termoptions = "<option value='$value1'"
        .($term === $value1 ? " selected='selected'>" : ">")."$label1</option>\r\n";
$termoptions .= "<option value='$value2'"
        .($term === $value2 ? " selected='selected'>" : ">")."$label2</option>\r\n";
$termoptions .= "<option value='$value3'"
        .($term === $value3 ? " selected='selected'>" : ">")."$label3</option>\r\n";
?>
<!-- 呈现所属部门在指定学期的开课信息列表 -->
<div style="margin: 20px 0 30px 0">
    如果需要添加开课信息，请单击<a href="opencourse_p.php?v=1">添加开课信息</a>
</div>
<form method="POST">
    <p>
    <label for="i1">选择学期：</label>
    <select id="i1" name="term"><?php echo $termoptions; ?></select>
```

```
    <input type="submit" name="submit1" value="确 认" />
    </p>
</form>
<?php
if(array_key_exists('ocs', $_SESSION)) {   // 呈现开课信息列表
?>
<style type="text/css">
    table .c1 {width: 300px; padding-left: 10px}
    table .c2 {width: 100px}
    table .c3 {width: 60px}
    table .c4 {width: 80px}
    table .c5 {width: 60px}
</style>
<form action="" method="POST" style="margin-top: 0px">
    <table>
    <thead>
      <tr>
      <td class="c1">课程</td><td class="c2">任课教师</td>
      <td class="c3">状态</td><td class="c4">选课人数</td><td class="c5"></td>
      </tr>
    </thead>
    <tbody>
      <?php
      $cnt1 = $cnt2 = $cnt3 = 0;
      for($i=0; $i<count($_SESSION['ocs']); $i++) { // 处理各开课信息
        $sstatus = ""; $del = "";
        $status = $_SESSION['ocs'][$i][3];
        if($status==='1') { $sstatus = '选课'; $cnt1++; }
        elseif($status==='2') { $sstatus = '教学'; $cnt2++; }
        else{ $sstatus = '结课'; $cnt3++; }
        // 为状态为 1 或 2 的开课信息，在其右端设置一个删除按钮
        if($status==='1' || $status==='2') {
          $url = $_SERVER['SCRIPT_NAME']."?v=2";
          $lid = $_SESSION['ocs'][$i][0];
          $del = "<form action='$url' method='POST' style='margin: 0'>"
                    . "<input type='hidden' name='lid' value='$lid'/>"
                    . "<input type='submit' class='text' value='删除'/></form>";
        }
        ?>
        <tr>
          <td class="c1"><?php echo $_SESSION['ocs'][$i][1] ?></td>
          <td class="c2"><?php echo $_SESSION['ocs'][$i][2] ?></td>
          <td class="c3"><?php echo $sstatus ?></td>
```

```
                <td class="c4"><?php echo $_SESSION['ocs'][$i][4] ?></td>
                <td class="c5"><?php echo $del ?></td>
            </tr>
        <?php } ?>
    </tbody>
    <tfoot>
        <tr style="height: 10px"><td colspan="4"></td></tr>
    </tfoot>
    </table>
</form>
<div>
[选课]状态：<?php echo $cnt1 ?>门课，[教学]状态：<?php echo $cnt2 ?>门课，
[结课]状态：<?php echo $cnt3 ?>门课。
</div>
<?php
if($cnt1>0) {  // 当有“选课”状态的开课信息时处理
    $url = $_SERVER['SCRIPT_NAME']."?v=2";
    $cs = "<form action='$url' method='POST' style='display: inline-block'> "
          . "<input type='hidden' name='cs' value='y'/>"
          . "<input type='submit' class='text' value='[选课]=>[教学]' /></form>";
    echo "<span>";
    echo "如果需要将其中处于[选课]状态的课程转成[教学]状态，";
    echo "请单击</span>".$cs;
}
}       // 开课信息列表呈现结束
?>
```

在“开课列表”模块文件 opencourse_list.php 代码中，多次调用 getOpenCourseData 函数，以从数据库中获取指定部门在指定学期的开课数据。下面是该函数的代码，它保存在函数库 mysql.php 文件中。该文件位于教务选课系统项目 xk 中的“源文件”结点下的 lib 子目录。代码如下：

```
<?php
/*
 *  功能：获取指定部门在指定学期的开课信息
 *  参数：$mysqli：对数据库 elective_manage 的连接对象
 *        $dept：登录管理员所属部门
 *        $term：学期
 */
function getOpenCourseData($mysqli, $dept, $term) {
    $query = "SELECT oc.lid, cname, tname, status, "
             ."(SELECT COUNT(*) FROM elective WHERE lid = oc.lid) num "
            ."FROM opencourse oc, course c, teacher t WHERE oc.cn = c.cn "
```

```
            . "AND oc.tn = t.tn AND dept = '$dept' AND term = '$term'";
    $result = executeSql($mysqli, $query);
    $ocs = array();
    while($row = $result->fetch_array()) {
      $oc=array($row['lid'],$row['cname'],$row['tname'], $row['status'],
     $row['num']);
      array_push($ocs, $oc);
    }
    return $ocs;
}
?>
```

习　题　10

1. 写出下面 PHP 代码运行的输出结果

（1）

```
$alpha = 'abcdefghijklmnopqrstuvwxyz';
$letters = array(15, 7, 15);
foreach($letters as $val) {
   echo $alpha[$val];
}
```

（2）假设之前没有定义$myarray 数组。

```
define("myvalue", "10");
$myarray[10] = "Dog";
$myarray[] = "Human";
$myarray["myvalue"] = "Cat";
$myarray["Dog"] = "Cat";
print $myarray[myvalue];
```

（3）假设之前没有定义$myarray 数组。

```
define("myvalue", "11");
$myarray[10] = "Dog";
$myarray[] = "Human";
$myarray["myvalue"] = "Cat";
$myarray["Dog"] = "Cat";
print $myarray[myvalue];
```

（4）

```
$myarray = array("aaa", "bbb", "ccc");
foreach ($myarray as $key => $value) {
   $myarray[$key] = $value."-123";
```

```
}
print $myarray[1];
```

(5)

```
$myarray = array("aaa", "bbb", "ccc");
foreach ($myarray as $value) {
   $value .= "-123";
}
print $myarray[1];
```

(6)

```
define("STOP_AT", 1024);
$result = array();
for($idx=1; $idx<STOP_AT; $idx*=2) {
   $result[] = $idx;
}
print_r($result[8]);
```

(7)

```
$fruits = array("orange", "banana", "apple", "raspberry");
echo array_shift($fruits), array_pop($fruits);
```

(8)

```
list($x, $y) = array("x" => "orange", 1 => "banana", 0 => "apple", "y" =>
"raspberry");
echo $x,$y;
```

(9)

```
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");
print_r($a + $b);
```

(10)

```
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");
print_r(array_merge($a, $b));
```

(11)

```
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");
print_r(array_merge_recursive($a, $b));
```

（12）

```
$arr1 = array("OH", "CA", "NY", "HI", "CT");
$arr2 = array("OH", "HI", "CA", "NE", "IA");
$arr3 = array("TX", "MD", "NY", "OH", "HI");
$result1 = array_diff($arr1, $arr2, $arr3);
print_r($result1);
```

（13）

```
$a = array(1, "hello", 1, "world", "hello");
$b = array_count_values($a);
print_r($b);
```

2. 根据要求写 PHP 代码

（1）测试数组$arr 中是否包含键为"x"的元素，若存在输出该元素的值。

（2）测试数组$arr 中是否包含值为"y"的元素，若存在输出该元素的键。

（3）按元素值对数组$arr 各元素进行字符串化排序。排序时，各元素的键和值保持关联。

（4）$books 是一个二维数组，存储了一些图书的书名、出版社和单价。

```
$books = array(array("数据库原理", "高等教育出版社", 36.50),
          array("操作系统", "清华大学出版社", 32.00),
          array("Java 程序设计", "电子工业出版社", 35.00)
    );
```

现在请按图书单价对各图书进行降序排序。

（5）对小于等于 100 的自然数随机排序，并将排序结果存放在数组$arr 中。

3. 简答题

（1）在 PHP 中，数组元素的键和值可以分别取什么类型的数据？

（2）简述函数 array_key_exists、in_array 和 array_search 的功能，并举例说明其用法。

（3）函数 sort、asort 和 ksort 有什么区别？它们分别在什么情况下使用？

第 11 章　PHP 面向对象程序设计

本章主题：

- 类与对象；
- 访问控制；
- 构造方法与析构方法；
- 静态类成员；
- 继承与方法覆盖；
- 抽象类与接口。

面向过程的程序设计侧重于对客观世界的实体的行为进行抽象，而把表示实体状态的属性置于一个被动、附属并相对分离的地位。面向对象的程序设计则把实体的属性和行为作为一个整体加以抽象。与面向过程的程序设计相比，面向对象程序设计无论在思维方式还是编程技术上都有很大区别。

本章首先介绍类、对象和封装性等概念、类的简单声明以及对象的创建和使用，然后依次介绍类成员的访问控制、构造方法与析构方法、静态类成员和类常量，最后介绍继承、方法覆盖和动态性、抽象类以及接口等内容。

11.1　类 与 对 象

本节首先介绍类和对象的概念，然后介绍如何定义类，以及如何基于类创建实例对象、如何访问对象的实例变量和调用对象的实例方法。

11.1.1　概念

在面向对象程序设计中，类和对象是两个最基本的概念。很好地理解这两个概念是学习和掌握面向对象程序设计的基础。

1. 对象

对象（Object）是对客观世界里的任何实体的抽象。被抽象的实体可以是具体的物，也可以指某些概念，例如一名学生、一台计算机、一门课、一个长方形等。

实体有属性和行为：属性表示实体的静态特征，所有属性的组合反映实体的状态；行为表示实体的动态特征，实体的行为可能会影响或改变实体的状态。客观世界里的任何实体往往都有丰富的属性和复杂的行为。抽象的目的是要从这些丰富和复杂的属性和行为中选择和提炼出为解决问题所需要的属性和方法。

对象是对客观世界实体进行抽象形成的软件模型，由数据和方法两部分组成。数据（变量、数组）对应于属性，用于表示对象的状态。方法是对行为的抽象，用于表示对象所具有的操作或所能够提供的服务。

对象是数据与方法的封装体。通过封装，对象可以对外界隐藏它的数据结构和方法的具体实现算法，而只把方法的格式信息（方法名、形参）和行为信息（功能）露在封装界面上。外界要使用一个对象，并不需要了解对象内部的实现细节，而只需知道对象封装界面上的信息，即该对象能够提供哪些服务。

2. 类

类（Class）是对一类相似对象的描述，这些对象具有相同的属性和行为、相同的变量（数据结构）和方法实现。类定义就是对这些变量和方法实现进行声明和描述。类好比是一类对象的模板，有了类定义后，就可以基于类生成所需的对象，称为类的实例。这里，类是一种数据类型，而类的一个实例对象是这种类型的一个数据。

因为类的每个实例共用相同的方法，所以在程序运行时，不管基于这个类产生了多少个实例，类中定义的每个方法的代码在内存中只需要一个副本。另一方面，类的各个实例虽然采用相同的变量来表示状态，但它们在变量上的取值完全可以不同。这些对象一般有着不同的状态，且彼此间相对独立。也就是说，类中定义的实例变量在内存中可能有多个副本，每个副本属于某个实例。

类应该提供对象封装的机制。在类的定义中，可以指定每个对象露在封装界面上的信息是什么、隐藏在封装界面里的内容是什么。这主要通过访问修饰符来实现的，用 public（公共的）修饰的变量和方法可以被外界访问或调用；用 private（私有的）修饰的变量和方法则不能被外界访问或调用。根据对象封装性的要求，实例变量一般用 private 修饰，而方法通常用 public 修饰。一般来说，这样定义的类也更能满足易修改、易维护、易重用的要求。

11.1.2 定义类

类的定义出现在 PHP 标签内。类的定义使用关键字 class，后面跟着类名，然后是类体。类体以左花括号开始、以右花括号结束，内含成员变量定义、构造方法、析构方法和成员方法定义。格式如下：

```
[final | abstract] class <类名> {
   [<成员变量定义>]*
   [<构造方法>]
   [<析构方法>]
   [<成员方法定义>]*
}
```

用关键字 final 修饰的类称为最终类，最终类是不能被扩展的。用关键字 abstract 修饰的类称为抽象类，抽象类通常会包含抽象方法，而且需要被扩展。一个类不能同时用关键字 final 和 abstract 修饰。

成员变量包括实例变量和静态变量，成员方法包括实例方法和静态方法。本节主要考虑用于存储对象状态的实例变量和用于表示对象行为的实例方法，关于静态变量、静态方法、构造方法以及析构方法将在后面各节分别介绍。

下面类定义的例子声明了一个名为 Fruit 表示水果的类。代码如下：

```
class Fruit {
  private $name = 'banana';
  private $color = 'yellow';
  function show() {
    echo 'a '.$this->color.' '.$this->name;
  }
}
```

该类定义了两个实例变量（$name 与$color）和一个实例方法（show）。与定义函数一样，定义方法也使用关键字 function。

类名和方法名都可以是任何非 PHP 保留字的合法标识符。一个合法类名和方法名以字母或下画线开头，后面跟着若干字母、数字或下画线。与函数名一样，类名和方法名也不区分大小写。

在 PHP 中，一个类中不允许出现两个同名的方法，即使它们有不同数目的形参。也就是说，PHP 不支持方法重载（overload）。但利用 PHP 中的魔术方法__call 可以产生方法重载的效果。

在类体内，要访问实例变量或调用实例方法，可以使用关键字$this 和对象成员访问运算符->，其一般格式如下：

```
$this-><实例变量名>   // 实例变量名不含$
$this-><实例方法名>([<实参表>])
```

关键字$this 表示对当前对象的引用。实例方法表示对象的行为，离开了对象，实例方法是没有意义的，也是无法调用的。实例方法必须通过对象调用，表示该对象的某种行为发生了。当通过某个对象调用一个实例方法时，该对象就称为当前对象。

在 Fruit 类中，实例方法 show 中的代码就通过此格式访问了实例变量$color 和$name。其中$this->color 和$this->name 分别表示访问当前对象的$color 变量值和$name 变量值。

11.1.3 创建和使用对象

一旦定义好了一个类，就可以使用实例创建表达式创建这个类的实例。实例创建表达式的一般格式如下：

```
new <类名>([<参数列表>])
```

其中，<参数列表>是一个可选项，如果要使用该选项，那么类定义中就需要有相应的构造方法。在没有介绍构造方法之前，可以暂时不考虑该选项。

实例创建表达式用于创建指定类的一个实例对象。其具体功能包括：

（1）为实例对象分配内存空间；

（2）创建并初始化实例变量；

（3）返回对该实例对象的一个引用。

下面代码创建 Fruit 类的一个实例对象，并把对该对象的引用赋给变量$aFruit，然后调用该对象的 show 方法：

```
$aFruit = new Fruit();
$aFruit->show();
```

对一个对象的引用值，除了可以判断其类型之外，并没有其他的操作可言。但是对对象的实例变量的访问以及对对象的实例方法的调用都需要通过该引用值进行。对包含对象引用值的变量（如$aFruit）,可称其为引用类型变量。

在类体外，要访问对象的实例变量或调用对象的实例方法，应通过对象引用、使用对象成员访问运算符->进行，其一般格式如下：

```
<引用类型变量名>-><实例变量名>  // 实例变量名不含$
<引用类型变量名>-><实例方法名>([<实参表>])
```

基于一个类可以根据需要创建任意数量的实例，每个实例拥有属于自己的实例变量。例如，下面代码创建了 Fruit 类的两个实例$fruitone 和$fruittwo，两个实例都有自己的实例变量$color 和$name，两者之间相互独立：

```
$fruitone = new Fruit();
$fruittwo = new Fruit();
```

可以使用运算符==、!=、===和!==对两个对象进行比较。当两个对象基于同一类创建且各实例变量的值相等，则两个对象是相等的（==），但两个对象并不全等（===）。只有当比较的两个对象的确是同一个对象时，全等比较（===）才成立。例如：

```
var_dump($fruitone==$fruittwo);    // true
var_dump($fruitone===$fruittwo);   // false
$tmp = $fruitone;
var_dump($fruitone==$tmp);         // true
var_dump($fruitone===$tmp);        // true
```

这里，变量$fruitone 和$fruittwo 引用两个不同的对象，但这两个对象都是 Fruit 类的实例，且各实例变量具有相同的取值。第 3 行代码把$fruitone 变量的引用值赋给$tmp 变量，这样两个变量具有相同的引用值，引用同一个对象。

【例 11-1】 定义一个表示长方形名为 Rectangle 的类，其中 getArea 方法计算并返回长方形的面积，getPerimeter 方法计算并返回长方形的周长。代码如下：

```
<?php
class Rectangle {
  private $width, $height;
  function set($w, $h) {
    $this->width = $w;
    $this->height = $h;
  }
  function getArea() {
    return $this->width * $this->height;
  }
```

```
11.    function getPerimeter() {
12.       return 2 * ($this->width + $this->height);
13.    }
14. }
15.
16. $r1 = new Rectangle();
17. $r1->set(10, 15);
18. $r2 = new Rectangle();
19. $r2->set(20, 12);
20. echo "area1=", $r1->getArea();
21. echo " perimeter1=", $r1->getPerimeter(), "</br />";
22. echo "area2=", $r2->getArea();
23. echo " perimeter2=", $r2->getPerimeter();
24. ?>
```

下面是代码运行的输出结果：

```
area1=150 perimeter1=50
area2=240 perimeter2=64
```

其中，第 2~14 行代码是对 Rectangle 类的定义，第 16~23 行代码演示了对该类的使用。

11.2 访 问 控 制

本节介绍访问修饰符以及魔术方法__get 和__set。访问修饰符用于指定成员变量或成员方法的访问级别，魔术方法__get 和__set 则可以对私有变量的读取和设置变得更加方便。

11.2.1 访问修饰符

在声明类成员时，可以指定访问修饰符控制其访问级别。在 PHP 中，访问修饰符有 3 个，分别定义了 3 种访问级别：

（1）private：私有的，仅能在所在类内被访问。

（2）protected：受保护的，能在所在类及其子类内被访问。

（3）public：公共的，能在所有位置被访问。

在 PHP 中，对实例变量来说，没有默认的访问级别。在声明实例变量时，必须选择其中一个访问修饰符修饰。对成员方法来说，默认的访问级别是公共的。也就是说，如果在声明成员方法没有指定访问修饰符，则默认为 public。

通常，用于修饰成员变量的各种修饰符应放置在变量名之前，但修饰符之间的次序无关紧要；用于修饰成员方法的各种修饰符应放置在关键字 function 之前，但修饰符之间的次序无关紧要。

为体现对象的封装性，实例变量通常被定义为私有的，而对外公开的实例方法通常被定义成公共的。

【例 11-2】 使用访问修饰符。代码如下：

```
<?php
class AccessModifier {
  private $a = 'aaa';
  public $b = 'bbb';
  private function getInfo() {
     return $this->a.'-'.$this->b;
  }
  public function showInfo() {
     echo $this->getInfo();
  }
}
$m = new AccessModifier();
echo $m->b, '<br/>';
echo $m->showInfo();
?>
```

下面是代码运行的输出结果：

```
bbb
aaa-bbb
```

在该例代码中，变量$a 和方法 getInfo 是私有的，只能在类体内被访问。变量$b 和方法 showInfo 是公共的，既可以在类体内被访问，也可以在类体外被访问。

可以利用 foreach 语句来遍历对象的实例变量。此时依赖于实例变量的可访问性，也就是，foreach 语句只能遍历对象中可访问的实例变量。

【例 11-3】 利用 foreach 语句遍历对象中的实例变量。代码如下：

```
<?php
class Traversal {
  private $a = 'aaa';
  public $b = 'bbb';
  function show() {
     foreach($this as $key=>$value) {
        echo $key.'-'.$value.'<br/>';
     }
  }
}
$m = new Traversal ();
$m->show();
echo '------------------<br/>';
foreach($m as $key=>$value) {
   echo $key.'-'.$value.'<br/>';
}
?>
```

下面是代码运行的输出结果：

```
a-aaa
b-bbb
------------------
b-bbb
```

该例代码中，成员方法 show 中的 foreach 语句可以遍历当前对象的所有实例变量，即$a 和$b。类体外的 foreach 语句只能遍历对象$m 中的公共实例变量，即$b。

11.2.2　魔术方法__get 和__set

为体现对象的封装性，在定义类时，实例变量通常被定义为私有的。但有些情况下，需要频繁访问实例变量，如果为每个私有的实例变量都定义相应的 get 方法和 set 方法，代码就会显得非常臃肿。此时可以考虑在类体内定义魔术方法__get 和__set。

当脚本代码试图读取对象的一个实例变量值时，如果该实例变量是不可访问（甚至是不存在）的，那么 PHP 系统将调用对象的__get 方法（如果存在）。__get 方法应该声明一个形参，该形参将接收正被访问的实例变量的名称。__get 方法体可以直接返回该实例变量的值，也可以进行适当的控制和处理。

当脚本代码试图设置对象的一个实例变量值时，如果该实例变量是不可访问（甚至是不存在）的，那么 PHP 系统将调用对象的__set 方法（如果存在）。__set 方法应该声明两个形参，第一个形参用于接收正要被设置的实例变量的名称，第二个形参用于接收要被设置的目标值。__set 方法体可以完成对指定实例变量的赋值。

【例 11-4】 使用魔术方法。代码如下：

```
<?php
class Magic {
  private $tn, $tname, $dept;
  function __set($propName, $propValue) {
     $this->$propName = $propValue;
  }
  function __get($propName) {
     $vars = array('tn', 'tname', 'dept');
     if(in_array($propName, $vars)) {
        return $this->$propName;
     } else {
        return NULL;
     }
  }
  function __toString() {
     return "职工(".$this->tn.", ".$this->tname.", ".$this->dept.")";
  }
}
$vars = array('tn', 'tname', 'dept');
```

```
20. $obj = new Magic();
21. $obj->$vars[0] = "0901"; $obj->$vars[1] = "刘绍军"; $obj->$vars[2] =
    "信息学院";
22. echo "职工号：", $obj->$vars[0], "<br />";
23. echo "姓名：", $obj->$vars[1], "<br />";
24. echo "所属部门：", $obj->$vars[2], "<br />";
25. echo "------------------------------------------<br />";
26. echo $obj;
27. ?>
```

下面是代码运行的输出结果：

```
职工号：0901
姓名：刘绍军
所属部门：信息学院
------------------------------------------
职工(0901，刘绍军，信息学院)
```

在该例中，实例变量$tn、$tname 和$dept 都是私有的，所以类体外是无法正常访问的。这里利用魔术方法__set 和__get 实现对这些变量的设置和读取。第 21 行代码通过隐含调用__set 方法完成对各实例变量的设置，第 22~24 行代码通过隐含调用__get 完成对各实例变量值的读取。

在第 21~24 行代码中，$vars[0]的值为'tn'，所以$obj->$vars[0]就相当于$obj->tn。类似地，$obj->$vars[1]相当于$obj->tname；$obj->$vars[2]相当于$obj->dept。

__toString()也是一个魔术方法，它会在当对象要被转换为字符串时被自动调用。该方法必须返回一个字符串，否则将发生一个致命（Fatal）错误信息。一般来说，__toString()方法应该返回一个能反映对象当前状态的字符串。

第 26 行代码输出变量$obj 的值，由于$obj 是一个对象，而不是一个字符串，所以它会被先转换成字符串，然后再输出。此时，PHP 系统将自动调用该对象的__toString()方法，并以该方法的返回值作为对象转换的结果。

在 PHP 中，魔术方法的方法名都是以__（两个下画线）开头的。开发人员在定义和命名类的成员方法时，一般不应以__为前缀。

11.3　构造方法与析构方法

当用 new 表达式创建类的实例时，若该类定义有一个构造方法，PHP 系统会自动调用类的构造方法。构造方法用以初始化实例对象的状态，即初始化实例变量。构造方法的方法名固定为__construct。

【例 11-5】 使用构造方法。代码如下：

```
<?php
class Complex {
  private $r, $i;
```

```
    function __construct($r, $i) {
      $this->r = $r;
      $this->i = $i;
    }
    function add($c) {
      return new Complex($this->r+$c->r, $this->i+$c->i);
    }
    function __toString() {
      return $this->r.'+'.$this->i.'i';
    }
}
$c1 = new Complex(1, 2);
$c2 = new Complex(3, 4);
$c3 = $c1->add($c2);
echo $c3;
?>
```

下面是代码运行的输出结果：

```
4+6i
```

该例的 Complex 类定义有构造方法，其中包含两个形参。这样，当用 new 表达式创建该类的实例时，也应该相应地提供两个实参。在创建实例的过程中，构造方法被自动调用，各实参会传递给相应的形参。如果实参的数目少于形参的数目，系统将给出警告（Warning）信息，未获得值的形参将是未定义的。如果实参的数目多于形参的数目，则多余的实参被忽略。

PHP 也提供析构方法的特性。与构造方法在创建实例时被自动调用相对应，析构方法在对象销毁前被自动调用。通常，一个对象在失去所有引用或脚本执行结束时被销毁。析构方法的固定名称为__destruct。

【例 11-6】 使用析构方法。代码如下：

```
<?php
class Destruct {
    function __construct() {  // 构造方法
      echo "对象正被创建<br/>";
    }
    function __destruct() {   // 析构方法
      echo "对象将被销毁<br/>";
    }
}
echo "start...<br/>";
$obj = new Destruct();
echo "processing...<br/>";
$obj = NULL;
echo "end!<br/>";
```

```
15. ?>
```

下面是代码运行的输出结果：

```
start...
对象正被创建
processing...
对象将被销毁
end!
```

第 11 行代码创建 Destruct 类的一个实例，其中的构造方法会被自动隐含调用。第 13 行代码将 NULL 值赋给变量$obj，使之前创建的对象失去了唯一对其的引用，此时，PHP 系统会自动隐含调用类中定义的析构方法。

11.4 静态类成员

这里介绍静态类成员，包括静态变量、静态方法以及类常量。

11.4.1 静态变量与静态方法

类的静态成员属于类。所谓“静态”是指这些变量和方法不由类的各实例专属，而由类的所有实例共享。

要把一个变量或方法定义成静态的，只需用关键字 static 修饰它们。默认情况下，静态成员都是公共的。

与实例成员不同，静态成员通过类名、使用范围解析符（::）进行访问。格式如下：

```
<类名>::<静态变量名>   // 静态变量名包括$
<类名>::<静态方法名>([<实参表>])
```

在所在类内，通常也可以通过关键字 self、使用范围解析符（::）进行访问。格式如下：

```
self::<静态变量名>    // 静态变量名包括$
self::<静态方法名>([<实参表>])
```

【例 11-7】 使用静态变量和静态方法。计算两个整数的最大公约数。代码如下：

```
<?php
class StaticDemo {
  static $m, $n;
  static function gcd() {
     $m = StaticDemo::$m;
     $n = StaticDemo::$n;
     $r = $m % $n;
     while($r!==0) {
        $m = $n;
        $n = $r;
```

```
11.         $r = $m % $n;
12.       }
13.       return $n;
14.    }
15. }
16. StaticDemo::$m = 20;
17. StaticDemo::$n = 8;
18. echo StaticDemo::$m, "%", StaticDemo::$n,  "=", StaticDemo::gcd();
19. ?>
```

下面是代码运行的输出结果：

```
20%8=4
```

在该例代码中，第 3 行定义的两个静态变量以及第 4 行开始定义的静态方法都没有用访问修饰符修饰。在默认情况下，它们的访问级别都是公共的。

类的静态成员属于类，但可以为该类的所有实例对象共享。通过实例对象或使用运算符->或::都可以访问静态方法。格式如下：

```
<引用类型变量名>-><静态方法名>([<实参表>])
<引用类型变量名>::<静态方法名>([<实参表>])
```

也可以通过实例对象、使用运算符::来访问静态变量，但不能通过实例对象或使用运算符->来访问静态变量。格式如下：

```
<引用类型变量名>::<静态变量名> // 静态变量名包含$
```

例如，在该例的最后添加以下两行代码，也会获得相同的结果。

```
$obj =new StaticDemo();
echo $obj::$m, "%", $obj::$n, "=", $obj->gcd();
```

实例变量和实例方法属于实例对象。没有实例对象，就不存在实例变量，实例方法也是没有意义的。静态变量和静态方法属于类。没有实例对象，这些静态变量和静态方法同样是可以访问和调用的。

一般来说，在实例方法体内，可以访问实例变量、调用其他的实例方法，也可以访问静态变量、调用静态方法；在静态方法体内，可以访问静态变量、调用其他的静态方法，但不能访问实例变量、调用实例方法。同样的道理，关键字$this 也不能用于静态方法体内。

11.4.2 类常量

利用关键字 const 可以把在类中始终保持不变的值定义为常量，称为类常量。类常量定义于类体内、方法体外。与在类体外的常量一样，在定义和使用类常量的时候，其名称不需要使用$符号。

在定义类常量时，不能使用关键字 static 和访问修饰符，但总是把它看作是静态和公共的。

【例 11-8】 使用类常量。代码如下：

```
1. <?php
2. class Constant {
3.    const SEC_PER_DAY = 60 * 60 * 24;
4.    function getDays() {
5.       return time()/self::SEC_PER_DAY;
6.    }
7. }
8. $obj = new Constant();
9. $days = (int)$obj->getDays();
10. echo Constant::SEC_PER_DAY, "<br />";
11. echo $days;
12. ?>
```

该例的 getDays 方法的功能是计算并返回自 1970 年 1 月 1 日至当前时间大约经历的天数。

该例第 3 行定义了一个名为 SEC_PER_DAY 的常量。因为它是静态的，所以应该使用运算符::访问。因为它是公共的，所以既可以在类内访问，也可以在类外访问。

第 5 行代码演示了如何在类体内通过关键字 self 访问类常量。第 10 行代码演示了如何在类体外通过类名访问类常量。

11.5 继　　承

继承是面向对象编程的基本机制，它允许对一个已定义的类进行扩展、派生出一个新的类。新类称为被扩展类的直接子类，被扩展类称为新类的直接超类。子类继承直接超类的成员变量和方法，并可以定义自己的成员变量和方法。

11.5.1 定义子类

在 PHP 中，继承和扩展机制通过 extends 短语实现，其语法格式如下：

```
[final | abstract] class <类名> extends <直接超类名> {
   ...
}
```

extends 短语指定了被扩展的类，称为当前定义类的直接超类，也称为当前定义类的父类，而当前定义类称为被扩展类的直接子类。通常把一个类 A 称为另一个类 C 的子类，是指满足下面条件之一者：

（1）类 A 是类 C 的直接子类。

（2）存在一个类 B，类 A 是类 B 的子类，类 B 是类 C 的子类。

如果类 A 是类 C 的子类，那么类 C 反过来被称为是类 A 的超类。一个类不能把自己作为超类。

在 PHP 中，继承只能是单重的，即 extends 短语只能指定一个直接超类。被指定的类必

须是非 final 的，一个 final 类不能被扩展，不能够有子类。

一个子类只能有一个直接超类，反过来，一个直接超类可以有许多子类。子类继承其直接超类中所有的非私有成员变量和成员方法。

可以把一个超类看作是一个存储着其所有子类的共同方法代码的仓库，这些共同的方法代码可以被其所有子类继承。这是面向对象程序设计中一个重要的代码重用机制。

【例 11-9】 子类和继承。代码如下：

```
<?php
class BaseClass {
  private $color;
  static $fruit = 'apple';
  function set($color) {
     $this->color = $color;
  }
  function get() {
     return $this->color;
  }
}
class ChildClass extends BaseClass {
  function show() {
     echo self::$fruit.'-'.$this->get();
  }
}
$obj = new ChildClass();
$obj->set('red');
$obj->show();
?>
```

下面是代码运行的输出结果：

```
apple-red
```

这里，子类 ChildClass 的成员包括：

（1）自身在类体内定义的实例方法 show；

（2）从父类 BaseClass 继承的静态变量$fruit、实例方法 set 和 get。

虽然实例变量$color 并不被子类继承，但当创建子类的实例时，该变量仍然会被创建。这样当调用子类实例的 set 方法，才可以为这个变量设置一个新的值。

一般来说，当创建类的一个实例时，该类及其所有超类中定义的实例变量都会被创建，而不管其访问级别如何。

在 PHP 中，子类不仅可以继承父类的成员变量和成员方法，也能够继承父类的构造方法和析构方法，只要：

（1）子类自身没有定义构造方法和析构方法；

（2）父类中的构造方法和析构方法是非私有的。

【例 11-10】 继承与构造方法。代码如下：

```
<?php
class Circle {
  private $radius;
  function __construct($radius) {
     $this->radius = $radius;
  }
  function getRadius() {
     return $this->radius;
  }
}
class Spheroid extends Circle {
  function getVolume() {
     return 4/3*M_PI*pow($this->getRadius(),3);
  }
}
$s = new Spheroid(10);
echo $s->getVolume();
?>
```

下面是代码运行的输出结果：

```
4188.7902047864
```

这里，子类 Spheroid 继承父类 Circle 中的构造方法。由于该构造方法带有一个形参，所以当用 new 表达式创建 Spheroid 类的实例时，也必须带一个实参。

11.5.2　方法覆盖

在 PHP 中，一个类中不允许有两个同名的成员方法。所谓方法覆盖是指在子类中定义了一个方法，该方法与父类中的一个方法具有相同名称，从而使父类中的那个方法不能被子类继承。这里，覆盖方法和被覆盖方法要么都是实例方法，要么都是静态方法，否则会产生一个致命（Fatal）的错误信息。

通常，方法覆盖是指实例方法的覆盖。如果说实例方法代表一类对象的某种行为，那么方法覆盖意味着：子类对象与父类对象相比，其某种行为已发生变化了。

当出现方法覆盖时，覆盖方法的访问级别应该与被覆盖方法的相同或更宽。如果被覆盖方法是 public 的，覆盖方法也必须是 public。如果被覆盖方法是 protected 的，覆盖方法可以是 protected 或 public。

一个被声明为 final 的方法称为最终方法。父类中最终方法可以被子类继承，但无法被子类覆盖。

【例 11-11】 方法覆盖。代码如下：

```
<?php
class BaseClass {
  function m($str) {
```

```
4.       echo $str.'<br/>';
5.     }
6.     function display() {
7.       $this->m("base");
8.     }
9. }
10. class SubClass extends BaseClass {
11.     function m($s) {
12.       parent::m($s);
13.       echo strlen($s).'<br/>';
14.     }
15.     function show() {
16.       $this->m('sub');
17.     }
18. }
19. $obj1 = new BaseClass();
20. $obj1->display();
21. echo "-------------<br />";
22. $obj2 = new SubClass();
23. $obj2->show();
24. echo "-------------<br />";
25. $obj2->display();
26. ?>
```

下面是代码运行的输出结果：

```
base
-------------
sub
3
-------------
base
4
```

该例的父类 BaseClass 声明了两个实例方法 m 和 display，子类 SubClass 也声明了两个实例方法 m 和 show。这里，方法 m 出现了覆盖的情况，它意味着子类对象的相应的行为与父类对象的是不同的。子类 SubClass 的成员除包括自身声明的实例方法 m 和 show，还包括从父类继承的实例方法 display。

在子类中，若要调用父类中被覆盖的方法，可以通过关键字 parent，使用运算符::来实现，如例子中的“parent::m($s);”。

第 19 行和第 20 行代码创建父类 BaseClass 的一个实例对象$obj1，然后调用该对象的实例方法 display。此时，第 7 行代码调用的显然是定义于父类的 m 方法。

第 22 行和第 23 行代码创建子类 SubClass 的一个实例对象$obj2，然后调用该对象的实例方法 show。此时，第 16 行代码调用的应该是定义于子类的 m 方法。

第 25 行代码调用对象$obj2 的实例方法 display。由于$obj2 是子类的实例而非父类的实例，所以在执行第 7 行代码时，将调用定义于子类的 m 方法，而不是定义于父类的 m 方法。

在这里，第 7 行代码对 m 方法的调用，如果是通过父类对象调用的，那么实际调用的是定义于父类的 m 方法；如果是通过子类对象调用的，那么实际调用的是定义于子类的 m 方法。这种特性称为多态性，即同样的操作（方法调用代码）可以有不同的行为（调用不同的方法）。

与一般的方法覆盖相比，构造方法与析构方法也有类似的情况：如果子类定义了构造方法和析构方法，那么超类中的构造方法和析构方法就不会被子类继承。一般来说，子类中的构造方法和超类中的构造方法不必有相同的形参。在子类中，可以通过关键字 parent、使用运算符::调用父类的构造方法和析构方法，如：

```
parent::__construct(); // 调用父类的构造方法
parent::__destruct();  // 调用父类的析构方法
```

一般来说，应该在每个类中定义构造方法，以初始化定义在该类中的实例变量。每个构造方法在初始化实例变量前，应该先调用其父类的构造方法。这样，当我们创建类的一个实例时，该类及其所有超类中的实例变量（无论是否是私有的、是否被子类继承）都会被创建并初始化，且超类中的实例变量先初始化，子类中的实例变量后初始化。

11.5.3 检测类型

这里介绍几个函数，可以检测对象的类，或检测两个类之间是否存在继承关系。

（1）get_class 函数。语法格式如下：

```
string get_class(object $object)。
```

函数返回指定对象$object 的类的名字。如果$object 不是一个对象，函数返回 false。

（2）get_parent_class。语法格式如下：

```
string get_parent_class(mixed $object)
```

函数用于返回指定对象$object 的父类名。如果$object 不是对象，或者该对象的类没有父类，函数返回 false。其中参数$object 也可以是表示类名的字符串，此时函数返回指定类的父类名。如果$object 不是类名，或者该类没有父类，函数返回 false。

（3）is_subclass_of 函数。语法格式如下：

```
bool is_subclass_of(mixed $object, string $class_name)
```

函数检测指定对象$object 是否是指定类$class_name 的子类的实例对象。如果是，函数返回 true；否则函数返回 false。如果要检测指定对象是否为指定类或指定类的子类的实例对象，可以使用运算符 instanceof。

该函数的参数$object 也可以是表示类名的字符串，此时函数检测指定类$object 是否为指定类$class_name 的子类。如果是，函数返回 true；否则函数返回 false。

【例 11-12】 使用检测函数。代码如下：

```
<?php
class C1 {}
class C2 extends C1 {}
class C3 extends C2 {}
$c1 = new C1();
$c2 = new C2();
$c3 = new C3();
var_dump(get_class($c1));
echo "<br />";
var_dump(get_parent_class($c2), get_parent_class("C2"));
echo "<br />";
var_dump(is_subclass_of($c1,"C1"), is_subclass_of($c2,"C1"),
is_subclass_of($c3,"C1"));
?>
```

下面是代码运行的输出结果：

```
string(2) "C1"
string(2) "C1" string(2) "C1"
bool(false) bool(true) bool(true)
```

11.6　抽象类和接口

抽象类可以包含已实现的成员方法，也可以包含未实现的抽象方法。接口只能包含未实现的抽象方法。一个类至多只能扩展一个类，但可以实现多个接口。

11.6.1　抽象类

一个用关键字 abstract 修饰的类称为抽象类。抽象类不能被实例化，即不能用 new 表达式创建抽象类的实例。一个抽象类通常会包含抽象方法。

抽象方法也用关键字 abstract 修饰。抽象方法只有方法头，没有实现代码，方法体只有分号（;)。任何一个类，如果它包含有抽象方法，那么这个类就必须被声明为抽象的。

通常，一个抽象类总是要被扩展产生子类，并由其适当的子类覆盖实现它的抽象方法。一个类既可以继承其超类中已经实现的方法，也可以继承其超类中没有实现的方法（即抽象方法)。一个类只有覆盖实现其超类中的所有抽象方法才能被定义成非抽象类，否则也只能被定义成抽象类。

抽象方法不能被定义为 private，因为私有方法不能被子类继承，所以也就无法被实现和覆盖。

【例 11-13】 使用抽象类。代码如下：

```
<?php
abstract class X {
  protected $a = 1;
```

```
  abstract function m1();
  abstract function m2();
}
abstract class Y extends X {
  protected $b = 10;
  function m1() { return $this->a + $this->b; }
}
class Z extends Y {
  private $c = 100;
  function m2() {return $this->a + $this->b + $this->c; }
}
$obj = new Z();
echo $obj->m1(), "<br />";
echo $obj->m2();
?>
```

下面是代码运行的输出结果：

```
11
111
```

这里，类 X 是一个抽象类，其中包含两个抽象方法 m1 和 m2。类 Y 是类 X 的子类，其覆盖实现了方法 m1，但没有覆盖实现抽象方法 m2，所以类 Y 只能是抽象类。类 Z 是类 Y 的子类，其覆盖实现了方法 m2，并继承了已经实现的方法 m1。由于类 Z 实现了从其超类中继承的所有抽象方法 m1 和 m2（m1 的实现从超类 Y 中继承，m2 的实现在类体内定义），所以它可以被定义为非抽象的。

【例 11-14】 定义一个抽象超类 Shape，其中包含两个抽象方法和一个非抽象方法。两个子类（Rectangle_1 和 Circle_1）都继承了抽象超类中的非抽象方法并覆盖实现了其中的抽象方法。代码如下：

```
<?php
abstract class Shape {
  abstract function getArea();
  function showArea() {
    echo "area=".$this->getArea()."<br/>";
  }
  abstract function resize($factor);
}
class Rectangle_1 extends Shape {
  private $width, $height;
  function __construct($width, $height) {
    $this->width = $width;
    $this->height = $height;
  }
  function getArea() {
```

```
        return $this->width*$this->height;
    }
    function resize($factor) {
        $this->width *= $factor; $this->height *= $factor;
    }
}
class Circle_1 extends Shape {
    private $radius;
    function __construct($radius) {
        $this->radius = $radius;
    }
    function getArea() {
        return M_PI*$this->radius*$this->radius;
    }
    function resize($factor) {
        $this->radius *= $factor;
    }
}

$rec = new Rectangle_1(10, 20);
$rec->showArea();
$cir = new Circle_1(10);
$cir->showArea();
$cir->resize(0.2);
$cir->showArea();
?>
```

下面是代码运行的输出结果：

```
area=200
area=314.15926535898
area=12.566370614359
```

与普通超类相同，抽象超类可以被看作是一个存储着其所有子类的共同方法代码的仓库，这些共同的方法代码可以被其所有子类继承。在该例中，类 Rectangle_1 和类 Circle_1 中具有相同代码的方法 showArea 被存放在了抽象超类 Shape 中。

与普通超类不同，抽象超类还可以被看作是对其所有子类的共同行为的描述。比如 Shape 类除了声明了一个非抽象方法 showArea，还声明了两个抽象方法：getArea 和 resize，这就迫使其所有非抽象子类必须提供对这两个抽象方法的实现和覆盖。

11.6.2 定义接口

接口和类都是引用类型。与定义类相比较，定义接口使用关键字 interface。接口中只能定义常量和没有实现的方法作为成员。接口定义的一般格式如下：

```
interface <接口名> [extends <直接超接口名表>] {
```

```
    [const <常量名>=<常量值>;]*
    [[public] function <方法名>(<形参表>);]*
}
```

接口中的常量类似于类常量，在定义时，不能使用关键字 static 和访问修饰符，但总是把它看作是静态和公共的。接口中定义的常量可以被子接口或实现接口的类继承。

接口中的方法本质上是抽象方法，但不需要用关键字 abstract 修饰。接口中的方法必须是公共的，关键字 public 可以写也可以省略。

一个接口可以有多个直接超接口，此时，各直接超接口名列在 extends 短语中。子接口继承各直接超接口的所有常量和方法，并可定义新的成员。

【例 11-15】 定义接口。代码如下：

```
<?php
interface I1{
   const EAST = 1;
   function m1();
}
interface I2 extends I1 {
   function m2();
}
?>
```

该例定义了两个接口：I1 和 I2。接口 I2 继承了接口 I1 拥有的成员，所以 I2 共包含 3 个成员：一个公共的常量 EAST；两个公共的抽象方法 m1 和 m2。

11.6.3 实现接口

与抽象类一样，接口不能实例化。接口可以被相关类实现，此时接口为超类型，相关类为子类型。一个类只能继承一个类，但可以实现多个接口，这些接口在 implements 短语中列出。格式如下：

```
[abstract] class <类名> [extends <直接超类名>] implements <直接超接口名表> {
   ...
}
```

如果一个类指定有超接口，那么它就应该为超接口中的所有抽象方法提供实现，覆盖这些方法，否则就只能声明为 abstract。提供实现的覆盖方法可以是在这个类的类体中定义的，也可以是从它的父类中继承而来的。

【例 11-16】 假设原有 Circle 和 Rectangle 两个类，它们都声明有 getArea 和 resize 两个实例方法。代码如下：

```
<?php
class Circle {
   private $radius;
   function __construct($radius) {
```

```
        $this->radius = $radius;
    }
    function getArea() {
        return M_PI*$this->radius*$this->radius;
    }
    function resize($factor) {
        $this->radius *= $factor;
    }
}
class Rectangle {
    private $width, $height;
    function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }
    function getArea() {
        return $this->width*$this->height;
    }
    function resize($factor) {
        $this->width *= $factor;
        $this->height *= $factor;
    }
}
?>
```

现在又定义了接口 IShape，其中声明了 getArea、showArea 和 resize 这 3 个抽象方法：

```
<?php
interface IShape {
    function getArea();
    function showArea();
    function resize($factor);
}
?>
```

定义 Circle_1 类和 Rectangle_1 类，分别实现 IShape 接口。代码如下：

```
<?php
class Circle_1 extends Circle implements IShape {
    function __construct($radius) {
        parent::__construct($radius);
    }
    function showArea() {
        echo "area=".$this->getArea()."<br/>";
    }
}
```

```
class Rectangle_1 extends Rectangle implements IShape {
   function __construct($width, $height) {
      parent::__construct($width, $height);
   }
   function showArea() {
      echo "area=".$this->getArea()."<br/>";
   }
}
$rec = new Rectangle_1(10, 20);
$rec->showArea();
$cir = new Circle_1(10);
$cir->showArea();
$cir->resize(0.2);
$cir->showArea();
?>
```

Circle_1 类和 Rectangle_1 类在实现 IShape 接口的同时，分别扩展 Circle 类和 Rectangle 类。这样，它们就可以用从超类中继承的 getArea 和 resize 方法来覆盖实现接口中相应的抽象方法。

接口描述了所有实现该接口的类的共同行为。当一个非抽象类声明实现一个接口时，那么这个类就必须要实现该接口和其所有超接口中声明的抽象方法。

习　题　11

1. 简答题

（1）什么是对象？什么是类？类和对象有什么关系？

（2）成员方法的默认访问级别是什么？静态变量的默认访问级别是什么？

（3）如何定义类常量？类常量的默认访问级别是什么？

（4）什么是方法覆盖？在子类中，如何访问直接超类中被覆盖的方法？

（5）什么是接口？在接口中，如何定义成员方法？

（6）举例说明魔术方法__get 和__set 的作用。

（7）举例说明魔术方法__toString 的作用。

（8）什么是构造方法？什么是析构方法？构造方法和析构方法的方法名分别是什么？

2. 编程题

（1）定义一个表示矩形的名为 Rectangle 的类。该类需定义相应的实例变量并实现以下构造方法和实例方法：

```
function __construct($w, $h);   // 将矩形的宽和高均设为 1
function getArea();             // 计算矩形的面积
function getPerimeter();        // 计算矩形的周长
function __toString();      // 以格式“Rectangle(w,h)”返回当前矩形的字符串表示
```

（2）定义 SavingsAccount 类。该类需定义一个静态类变量$annualInterestRate 以存储年

利率、一个实例变量$savingsBalance 以存储储户当前的储蓄额，并实现以下方法：

```
function __connstruct($saving);      // 构造方法，初始化储蓄额
function calculateInterest($n);      // 计算当前储户$n 年的利息并加到储蓄额中
static function modifyInterestRate($rate);//为$annualInterestRate 设置新值
```

（3）假设已定义有表示圆的名为 Circle 的类，代码如下：

```
class Circle {
   private $radius;
   function __construct($r) {         // 构造方法，设圆的半径设为 r
      $this->radius = $r;
   }
   function getArea() {               // 计算圆的面积
      return pi()*$this->radius**2;  // 系统内置函数 pi()返回圆周率
   }
   function getPerimeter() {          // 计算圆的周长
      return 2*pi()*$this->radius;
   }
}
```

现在定义一个表示圆柱体的名为 Cylinder 的类，该类应该充分利用现有的 Circle 类，并实现以下构造方法和实例方法，代码如下：

```
function __construct($r, $h);        // 设置圆柱的底圆半径及高
function getArea();                   // 计算圆柱表面积
function getVolume();                 // 计算圆柱体积
```

注意：不要修改 Circle 类的定义。

（4）假设已经定义有一个 A 类。代码如下：

```
class A {
   private $x;
   function setX($x) {
      $this->x = $x;
   }
   function method() {
      return $this->x**2;            // 返回实例变量$x 的平方
   }
}
```

现在完善下面 MyClass 类的定义，按注释指定的要求实现其中 method 方法。代码如下：

```
class MyClass extends A{
   private $y;
   function setY($y) {
      $this->y = $y;
   }
```

```
    function method() {
        // 返回实例变量$x 的平方与实例变量$y 的平方的和
    }
}
```

注意：不要修改类的其他代码，也不要修改 MyClass 类中 method 方法的方法头。

（5）充电站可以为电动汽车充电。充电时间以小时为单位，充电的费用在一天的不同时间点（以小时为单位）是不同的。BatteryCharger 类包含一个实例变量和两个实例方法，代码如下：

```
class BatteryCharger {
    private $rateTable = array(50, 60, 60, 60, 70, 80, 90, 100, 90, 90, 90,
                               100, 100, 100, 90, 90, 90, 100, 100, 90, 90, 80, 70,
                                60);
    function getChargingCost($startHour, $chargeTime) {
        // 待实现
    }
    function getChargeStartTime($chargeTime) {
        // 待实现
    }
}
```

实例变量$rateTable 保存了不同时间点的充电费用(共 24 个数据)，如 0 点的费用是 50，1 点的费用是 60，…，23 点的费用是 60 等。注意，这些数据是可以改变的。

方法 getChargingCost 计算并返回从时间点$startHour 开始充电，连续充电$chargeTime 小时所需费用。

方法 getChargeStartTime 确定并返回一个起始充电时间点，以便连续充电$chargeTime 小时所需费用最低。若有两个起始充电时间点都能满足最低费用的要求，返回较小的起始充电时间点。

请实现上述两个方法。

第 12 章　Ajax 与 jQuery

本章主题：

- Ajax 基础；
- jQuery 相关概念；
- jQuery 选择器；
- jQuery 操作 HTML 元素；
- jQuery 事件处理；
- jQuery 动画效果；
- jQuery 中的 Ajax。

Ajax 是一种开发 Web 应用的方法，其思想是尽可能在客户端处理用户与应用的交互事件，如果有必要，就在后台采用异步的方式与服务器进行数据交换，并局部更新页面，从而提高应用的用户体验。jQuery 是一种 JavaScript 库，它允许开发人员以更便捷的方式开发高质量的客户端程序，也为实现 Ajax 方法的思想提供了有力的技术保证。

本章首先介绍 Ajax 的概念及基本实现技术，然后较为详细地介绍 jQuery 的相关概念与功能，包括 jQuery 对象、选中元素、jQuery 选择器、操作 HTML 元素、事件处理、动画效果以及 jQuery 对 Ajax 的支持等。

12.1　Ajax 基础

Ajax 是 2005 年由 Jesse James Garrett 提出的一种新的 Web 应用开发方法，它能够极大地改善 Web 应用的用户体验。

12.1.1　什么是 Ajax

Ajax（Asynchronous JavaScript And XML，异步 JavaScript 和 XML）不是一种新的编程语言或技术，而是现有的几种技术的一种结合。它产生了一种新的、强大的开发 Web 应用的方法。Ajax 方法的基本思想如下：

（1）基于 HTML+CSS 来呈现信息。

（2）使用 XMLHttpRequest 从 Web 服务器异步获取数据。

（3）利用 XML、JSON 等来表示数据。

（4）通过 DOM 实现页面的局部更新及动态效果。

（5）运用 JavaScript 将各个方面绑定在一起。

图 12-1 比较了传统的 Web 应用模型与 Ajax Web 应用模型。传统的 Web 应用模型建立在万维网基本结构的基础上，其工作过程是，用户通过与界面交互（例如单击超链接、提交表单等）触发送往服务器的 HTTP 请求；服务器接收到请求后，经过相应的处理、动态

产生 Web 页面，然后产生 HTTP 响应送往客户端；客户端浏览器接收到 HTTP 响应后，读取 Web 页面代码，经过解析和呈现，形成新的用户界面。从触发 HTTP 请求到新的用户界面的形成，用户一般什么都做不了，只有等待。

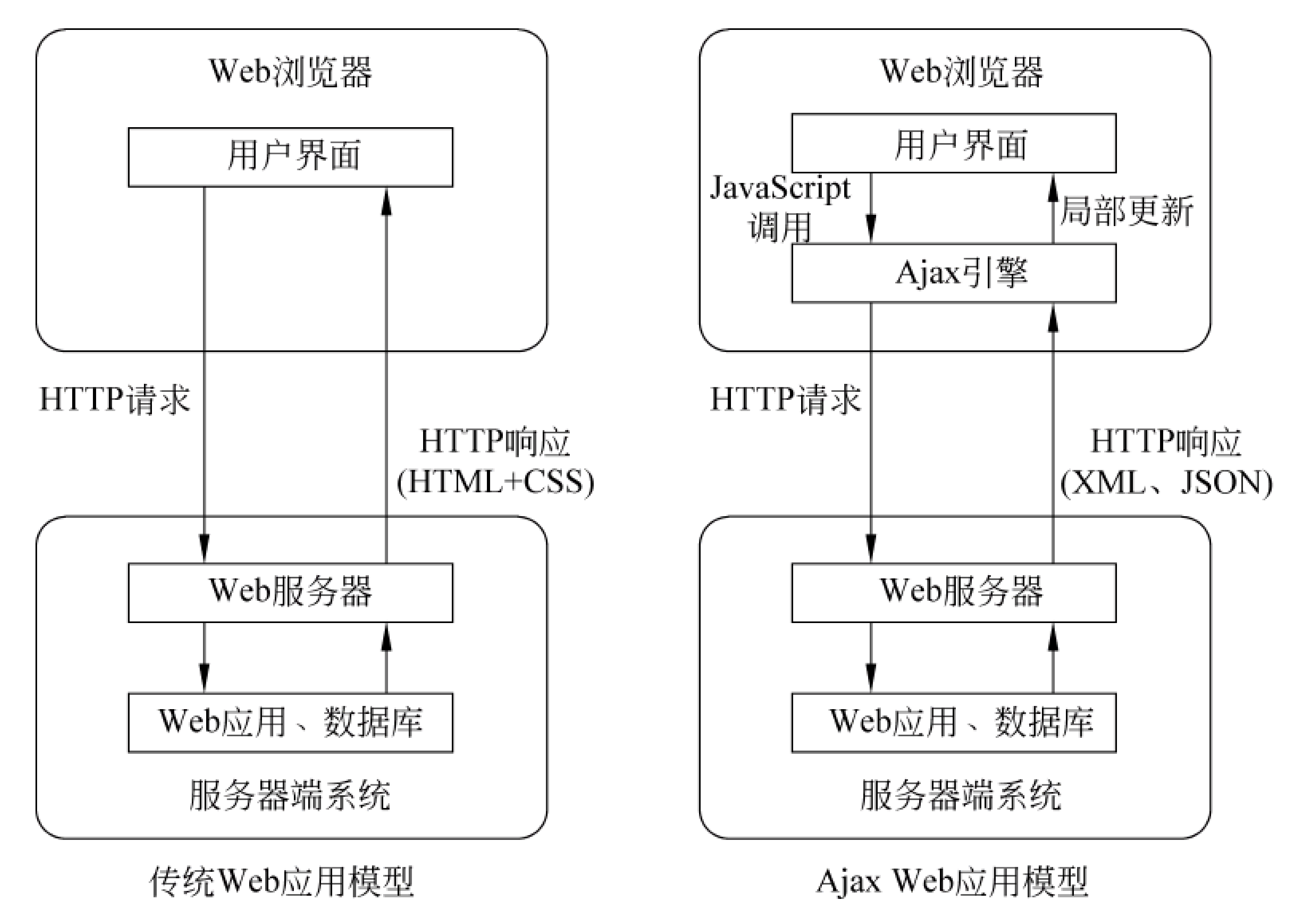


图 12-1　Ajax Web 应用模型与传统 Web 应用模型

与传统的 Web 应用模型相比，Ajax Web 应用模型在用户界面与服务器之间引入了一个新的部件，称为 Ajax 引擎。Ajax 引擎由 JavaScript 编写，负责处理用户与 Web 应用之间的交互。如果用户的交互动作只是需要对表单数据作初步的验证，或者要求访问的数据已经保存在客户端，那么就可以由 Ajax 引擎在客户端直接处理。如果用户的交互动作的确需要从服务器端获取数据，那么可以委托 Ajax 引擎与服务器进行通信。Ajax 引擎在后台向服务器发送异步的 HTTP 请求；服务器接收到请求后，经过相应的处理、动态产生处理结果（XML、JSON 等），然后产生 HTTP 响应送往客户端；Ajax 引擎接收到 HTTP 响应后，完成对响应的具体处理，例如实现在页面的局部更新。由于在后台触发异步的 HTTP 请求，所以在服务器处理期间，并不妨碍用户继续与界面进行交互，甚至再次发送 HTTP 请求。

利用 Ajax 方法，Web 应用可以在后台异步地向服务器发送数据和获取数据，而不会妨碍原有页面的显示和交互行为；Web 应用能够动态地局部更新原有的页面，而不需要完整地重载一个页面。这样的特点可以极大地改善 Web 应用的用户体验。一般来说，Ajax Web 应用主要有以下优点：

（1）改变了“点击-等待-页面刷新”模式，带来持续、动态的用户体验。

（2）减轻网络通信的负担。当需要对页面进行局部更新时，不需要完整地重载一个页面，而只需要从服务器端获取相应的数据。

（3）减轻服务器的负担。对用户交互的有些动作，可以在客户端直接处理，而不必由服务器“事必躬亲”。

（4）创建桌面应用风格的界面。引入存在于客户端的 Ajax 引擎来处理用户与 Web 应用之间的交互，使得交互动作的种类和响应速度都更接近于桌面应用界面的风格。

（5）促进页面模板和页面数据的分离。一个动态页面往往由静态的页面模板和动态的页面数据两部分组成。模板规定了数据的呈现格式，通常是不变的，而数据则需要根据应用的当前状态或用户提供的查询参数动态确定。Ajax 方法要求对这两部分内容有更清晰的分隔。

Ajax 方法改变了万维网固有的工作流程，由此也会带来一些问题。Ajax Web 应用的主要缺点包括：

（1）无法正常使用“后退”按钮的功能。浏览器只对由它自身装载的完整页面有历史记录，而对由 Ajax 引擎引起的页面局部更新状态是不会有历史记录的，所以希望通过单击“后退”按钮取消用户前一次的交互动作是行不通的。类似的道理，“前进”按钮、“刷新”按钮以及书签化等功能通常也是不能正常使用的。

（2）代码保密性差。由于使用 Ajax 方法的 JavaScript 脚本代码会随 HTML 文档下载到客户端保存和运行，不利于代码的保密。

（3）兼容性不理想。由于使用 Ajax 方法的 JavaScript 脚本代码是在客户端浏览器中运行的，而不同的浏览器对有关 JavaScript 代码的支持程度有所不同，所以编程时还需要考虑浏览器的类型。

12.1.2　XHR 对象

目前，XMLHTTPRequest 对象是使用 Ajax 方法的技术基础，简称 XHR 对象。XHR 对象是 JavaScript 本身支持的，它允许浏览器在后台与 Web 服务器交换数据，从而可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

1. 创建 XHR 对象

IE、Firefox、Chrome、Safari 以及 Opera 等所有的现代浏览器均支持 XHR 对象，其中 IE 浏览器把 XHR 对象实现为一个 ActiveXObject 对象。

创建一个 XHR 对象的代码因浏览器实现方式的不同而有所区别。对于非 IE 浏览器，一般采用下面代码创建 XHR 对象：

```
xhr = new XMLHttpRequest();
```

对于早期的 IE 浏览器，可以使用下面代码：

```
xhr = new ActiveXObject("Microsoft.XMLHTTP");
```

而对于 IE6 及之后版本的 IE 浏览器，则可使用下面代码：

```
xhr = new ActiveXObject("Msxml2.XMLHTTP");
```

2. 向服务器发送请求

要向 Web 服务器发送 HTTP 请求，需要调用 XHR 对象的 open 和 send 方法。

open 方法用于初始化 XHR 对象，指定 HTTP 请求的方法、URL 以及是否异步处理请求。格式如下：

```
open(<method>, <url>, <async>)
```

其中参数如下。

- <method>：指定 HTTP 请求的方法，如 GET、POST 等。
- <url>：被请求资源的 URL。通常采用相对 URL，即相对于包含该脚本的当前资源所在位置的地址。
- <async>：指定是异步处理（true）还是同步处理（false）当前请求，默认值为 true。在 Ajax 方法中，通常采用异步请求。

在调用 open 方法后，XHR 对象将其 readyState 属性设置为 1，表示处于“打开”状态。

在通过调用 open 方法准备好一个请求之后，可以调用 send 方法把该请求发送到服务器。格式如下：

```
send([<content>])
```

方法使用一个可选的参数<content>。在发送 POST 请求时，应该将该方法参数设置为当前请求所需的请求参数。当发送 GET 请求时，通常不用该参数，或者使用 null 参数调用该方法。

在调用 send 方法后，XHR 对象将其 readyState 属性设置为 2，表示处于“发送”状态。

例如，下面代码将发送一个 HTTP GET 请求：

```
xhr.open("GET", "demo_get2.php?fname=Bill&lname=Gates", true);
xhr.send(null);
```

又例如，下面代码将发送一个 HTTP POST 请求：

```
xhr.open("POST","ajax_test.asp",true);
xhr.setRequestHeader("Content-type",
                     "application/x-www-form-urlencoded");
xhr.send("fname=Bill&lname=Gates");
```

其中，setRequestHeader 方法为请求头设置一个域，第 1 个参数指定域名，第 2 个参数指定域值。该方法只能在调用 open 方法之后、调用 send 方法之前调用。

上述两个例子都是异步请求，因在调用 open 方法时<async>参数都被设置为 true。此时发送请求后，send 方法立即返回，后续脚本代码会继续执行。对于异步请求，一般需要监听 XHR 对象的 readystatechange 事件并注册相应的事件处理函数，以便在响应返回时能够及时处理。

如果是同步请求，即在调用 open 方法时<async>参数被设置为 false，那么发送请求后，send 方法将阻塞，直到服务器的响应返回。一旦 send 方法返回，可以访问 XHR 对象的 status 属性了解响应状态，访问 XHR 对象的 responseText 或 responseXML 属性获取响应内容。

3. readystatechange 事件

XHR 对象的 readyState 属性记录着当前请求-响应过程的状态信息，即所处的阶段，其值取 0~4 中的一个整型值，分别表示 5 种状态。

（1）0：“未初始化”状态。XHR 对象刚刚创建，但发送 HTTP 请求的各参数还未设置。

（2）1：“打开”状态。已调用 XHR 对象的 open 方法，发送 HTTP 请求的各参数已被设置。

（3）2：“已发送”状态。已调用 XHR 对象的 send 方法，但还没有收到响应。

（4）3：“正在接收”状态。已经接收到 HTTP 响应的头部信息，但响应体部分还没有完全接收结束。

（5）4：“已加载”状态。HTTP 响应已经完全被接收，responseText 或 responseXML 属性已被设置。

每当 readyState 属性值改变时，就会在 XHR 对象上触发 readystatechange 事件。如果发送的是异步请求，一般需要在发送请求前，先在 XHR 对象上注册一个相应的事件处理函数，以便监听和处理该事件。

通过设置 XHR 对象的 onreadystatechange 属性，可以为 readystatechange 事件注册相应的事件处理函数。可以事先编写事件处理函数，然后将 onreadystatechange 属性设置为该函数名；也可以直接将 onreadystatechange 属性设置为一个匿名函数，作为事件处理函数。

```
xhr.onreadystatechange = <function_name>;
```

或

```
xhr.onreadystatechange = function() {
...
}
```

在事件处理函数中，可以了解整个请求-响应过程所处的阶段、HTTP 响应的状态以及响应的内容，进而进行所需的处理。

4. 服务器响应

当 XHR 对象的 readyState 属性值为 3 或 4 时，可以访问 XHR 对象的 status 属性了解响应状态。当 readyState 属性值为 4 时，可以访问 XHR 对象的 responseText 或 responseXML 属性获取响应内容。

（1）status 属性：包含 HTTP 响应的状态码。如 200 表示成功处理，404 表示“未找到页面”错误等。

（2）responseText 属性：包含客户端从服务器端接收到的字符串形式的响应内容。

当 readyState 属性值为 0、1 或 2 时，该属性值为空串；当 readyState 属性值为 3 时，表示正在接收，responseText 属性还不可用；当 readyState 属性值为 4 时，该属性包含完整的响应内容。

（3）responseXML 属性：包含表示 XML 文档的 DOM 文档树。

如果来自服务器的响应内容是 XML 文档（HTTP 响应的 Content-Type 域应该指定为 text/XML 或 application/XML），则可以访问该属性获得一个相应的 DOM 文档树。然后通过 DOM API，可以读取 XML 文档中的数据。

无论何时，只要 readyState 属性值不为 4，则该属性的值为 null。如果接收到的 HTTP 响应内容不是 XML 文档，或者文档不能被正确分析（如文档结构不完整等），该属性的值也为 null。

【例 12-1】 录入提示。当用户在文本域中输入一个单词时，采用 Ajax 方法，从服务器实时获得提示信息并显示。

该例结果包含 3 个文件，首先给出页面文件 12-1.html 的代码，如下所示：

```
<!DOCTYPE html>
<html>
  <head>
    <title>12-1</title>
    <meta charset="UTF-8">
    <script src="clienthint.js"></script>
  </head>
  <body>
    <form>
       输入单词：
       <input type="text" id="txt1" onkeyup="showHint(this.value)" />
     </form>
     <p>提示：<span id="txtHint"></span></p>
   </body>
</html>
```

其中，showHint(this.value)是在文本域上为 keyup 事件注册的事件处理函数，传入的是文本域的当前值，该函数定义在 clienthint.js 文件中。下面是 clienthint.js 文件的代码：

```
var xhr = getXhr();
// keyup 事件的处理函数
function showHint(pre) {              //若文本框为空，无提示信息
  if (pre.length === 0) {
    document.getElementById("txtHint").innerHTML = "";
    return;
  }
  if (xhr === null) {                 //若创建 XHR 对象失败，显示错误信息
    alert ("Browser does not support HTTP Request");
     return;
   }
   var url = "gethint.php";
   url = url + "?pre=" + pre;
   xhr.onreadystatechange = function(){//为 readystatechange 事件注册处理函数
     if (xhr.readyState===4 && xhr.status===200) {
        // 用响应内容设置提示信息
        document.getElementById("txtHint").innerHTML=xhr.responseText;
     }
   };
   xhr.open("GET", url, true);
   xhr.send(null);                       //发送异步的 HTTP GET 请求
}
// 获得 XHR 对象
function getXhr() {
   try {
     xhr = new XMLHttpRequest();
```

```
   } catch (e) {
      try {
         xhr = new ActiveXObject("Msxml2.XMLHTTP");
      } catch (e) {
         xhr = new ActiveXObject("Microsoft.XMLHTTP");
      }
   }
   return xhr;
}
```

当执行 showHint 函数时，如果文本域非空，就会向服务器发送异步的 HTTP GET 请求，目标资源是 PHP 文件 gethint.php，请求参数为文本域的当前值。下面是 gethint.php 文件的代码：

```
<?php
// 创建单词库
$words[]="advice";
$words[]="affect";
$words[]="bamboo";
$words[]="classic";
$words[]="close";
// 获得请求参数 pre，即文本域的当前值
$pre = $_GET["pre"];
$hint = "";
// 若 pre 不为空，获取前部与 pre 相匹配的所有单词
if (strlen($pre) > 0) {
   for($i=0; $i<count($words); $i++) {
      if (strtolower($pre) === strtolower(substr($words[$i], 0,
      strlen($pre)))) {
         if ($hint=="") {
            $hint=$words[$i];
         } else {
            $hint=$hint.", ".$words[$i];
         }
      }
   }
}
// 确定并输出响应内容
if ($hint == "") {
   $response="no suggestion";
} else {
   $response=$hint;
}
echo $response;
?>
```

该 PHP 文件根据文本域的当前值，产生相应的提示信息并作为响应内容返回。

12.2　初识 jQuery

本节将介绍 jQuery 的一些基本情况，包括 jQuery 库、jQuery 对象和 jQuery 方法等概念，以及 jQuery 所能够提供的功能和代码特点等。

12.2.1　简介

jQuery 是由美国人 John Resig 于 2006 年创建的一个开源项目，目前已发展为最流行和广泛采用的 JavaScript 代码库，其主旨是“写的更少，但做的更多”。借助于 jQuery，开发人员能够以更加简捷的方式编写更易阅读的 JavaScript 代码，以实现特定的功能。jQuery 涉及的基本功能包括：

- HTML 元素选取；
- HTML 元素操作；
- CSS 操作；
- 页面事件处理；
- HTML 元素的动画效果；
- Ajax 等。

jQuery 是一个开源项目，可以从 jquery.com 网站免费下载。共有两个版本的 jQuery 可供下载：一份是精简过的（文件名带 min），另一份是未压缩的（供调试或阅读）。jQuery 库位于一个 JavaScript 文件中，其中包含了 jQuery 的所有函数和方法。要使用 jQuery，无须安装任何软件，只需通过<script>元素在 Web 页面中引入相应的 JavaScript 文件即可。代码类似如下：

```
<script src="jquery-3.1.0.min.js"></script>
```

下面通过一个简单的例子，感受一下 jQuery 代码的风格及其所能完成的功能的特点。

【例 12-2】 在页面中引入 jQuery 库，然后使用 jQuery 函数和方法，在页面加载时动态设置<div>元素的内容。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
     <title>12-2</title>
     <meta charset="UTF-8">
     <script src="jquery-3.1.0.min.js"></script>
     <script>
        $(document).ready(function() {
           $("div").html("<h2>欢迎使用 jQuery</h2>");
         });
      </script>
```

```
12.    </head>
13.    <body>
14.       <div></div>
15.    </body>
16. </html>
```

当访问该页面时，浏览器窗口内会以二级标题格式显示“欢迎使用 jQuery”字样。

在代码中，第 6 行代码引入 jQuery 库，第 8 行至第 10 行是 jQuery 代码。其中$()也就是 jQuery()函数，$是 jQuery 的别名，所以代码中也完全可以用 jQuery 替换$。jQuery 函数接收的参数可以是 document（文档对象）、window（窗口对象）或某个 DOM 元素对象，但更多的情况是一个表示选择器的字符串，如$("div")。jQuery 函数返回一个 jQuery 对象，下一小节将对此作详细介绍。

ready()和 html()是两个 jQuery 方法。这里，html()方法用于设置<div>元素的内容。下面介绍 ready()方法的作用。

在 jQuery 中，当一个 HTML 文档载入到客户端并创建相应的 DOM 树后，会产生一个文档准备就绪（ready）事件。可以在文档对象上用 ready 方法为该事件注册一个事件处理函数（通常是一个匿名函数），其代码如下：

```
$(document).ready(function(){
   ...
});
```

这样，一旦引发 ready 事件，传入 ready 方法的匿名函数就会立即执行。由于 ready 方法仅能通过包含文档对象的 jQuery 对象调用，所以上面代码也可以简写成如下格式，两者的功能是等价的。

```
$(function(){
   ...
});
```

综上所述，该例 jQuery 代码的功能是，当页面文档载入到客户端并创建相应的 DOM 树后，页面中<div>元素的内容被设置为“<h2>欢迎使用 jQuery</h2>”。

也可以为文档的 ready 事件注册多个处理函数，即在一个 HTML 文档中，上述代码可以出现多次。当文档的 ready 事件引发时，这些事件处理函数将按其注册的先后顺序依次被执行。

12.2.2 jQuery 对象

在客户端浏览器，一个 HTML 文档被表示为一个 DOM（Document Object Model，文档对象模型）树，树的根结点是 Document 对象。文档中的每个 HTML 元素被表示成树上的一个结点，称为 DOM 对象或 DOM 元素。JavaScript 为 DOM 对象定义一系列方法和属性，可供程序员调用和访问。

jQuery 对象是由 jQuery()函数（也就是$()）返回的对象。一个 jQuery 对象可以包含一组 DOM 元素。一个 jQuery 对象所包含的 DOM 元素也经常被称为选中元素。jQuery 为 jQuery

对象定义了一系列方法，即 jQuery 方法。

调用 jQuery 对象的 length 属性可以返回该 jQuery 对象包含的 DOM 元素个数。调用 jQuery 对象的 get(<index>)方法可以返回该 jQuery 对象中指定位置上 DOM 元素，其中索引值 index 从 0 开始计算。调用 jQuery()函数可以将一个 DOM 元素封装成一个 jQuery 对象。下面是这几个方法的使用：

```
var jqo1 = $("p");          // 获得一个包含所有段落元素的 jQuery 对象
var n = jqo1.length;        // 获取 jQuery 对象中包含的 DOM 元素个数
var dom_p = jqo1.get(n-1); // 获取 jQuery 对象中最后一个 DOM 元素
var jqo2 = $(dom_p);        // 将一个 DOM 元素转换成一个 jQuery 对象
```

下面是 jQuery 对象的一组方法，它们都会返回一个新的 jQuery 对象。新的 jQuery 对象或者包含原来 jQuery 对象中的特定 DOM 元素，或者包含与原来 jQuery 对象中的 DOM 元素有上下文关系的 DOM 元素。

（1）first()。构建一个新的 jQuery 对象返回，该 jQuery 对象仅包含当前 jQuery 对象中的第 1 个元素（索引值为 0）。

（2）last()。构建一个新的 jQuery 对象返回，该 jQuery 对象仅包含当前 jQuery 对象中的最后一个元素。

（3）eq(<index>)。构建一个新的 jQuery 对象返回，该 jQuery 对象仅包含当前 jQuery 对象中指定位置上的元素。

（4）children([<selector>])。构建一个新的 jQuery 对象返回，该 jQuery 对象包含当前 jQuery 对象中每个元素的子元素。如果指定参数选择器，则仅包含匹配选择器的子元素。

（5）parent([<selector>])。构建一个新的 jQuery 对象返回，该 jQuery 对象包含当前 jQuery 对象中每个元素的父元素。如果指定参数选择器，则仅包含匹配选择器的父元素。

（6）next([<selector>])。构建一个新的 jQuery 对象返回，该 jQuery 对象包含当前 jQuery 对象中每个元素的后面一个兄弟元素。如果指定参数选择器，则仅包含匹配选择器的后面一个兄弟元素。

（7）prev([<selector>])。构建一个新的 jQuery 对象返回，该 jQuery 对象包含当前 jQuery 对象中每个元素的前面一个兄弟元素。如果指定参数选择器，则仅包含匹配选择器的前面一个兄弟元素。

（8）siblings([<selector>])。构建一个新的 jQuery 对象返回，该 jQuery 对象包含当前 jQuery 对象中每个元素的所有兄弟元素。如果指定参数选择器，则仅包含匹配选择器的兄弟元素。

【例 12-3】 调用 jQuery 对象的方法。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
     <title>12-3</title>
     <meta charset="UTF-8">
     <script src="jquery-3.1.0.min.js"></script>
     <script>
```

```
8.         $(function() {
9.           $("p").css("font-weight", "700");
10.            $("p").next().css("background-color", "gray");
11.          });
12.       </script>
13.    </head>
14.    <body>
15.       <p>段落一</p>
16.       <div>内容一</div>
17.       <p>段落二</p>
18.       <div>内容二</div>
19.    </body>
20. </html>
```

在第 9 行代码中，jQuery 函数$("p")返回一个 jQuery 对象，其选中元素为两个段落元素。调用该 jQuery 对象的 css 方法，可以使两个段落元素的文字以粗体显示。第 10 行代码调用 next 方法，返回一个新的 jQuery 对象，包括所有段落元素的后继兄弟元素，即其选中元素为两个<div>元素。最后调用新的 jQuery 对象的 css 方法，使两个<div>元素以灰色背景显示。关于 css 方法，本书 12.4 节会做进一步介绍。

jQuery 库包括一个 jQuery 函数、一组 jQuery 方法及一系列的 jQuery 工具函数。jQuery 函数返回一个 jQuery 对象，jQuery 方法是 jQuery 对象所具有的方法。在很多情况下，jQuery 函数需要一个表示选择器的字符串作为参数。接下来首先介绍 jQuery 选择器，然后介绍常用的 jQuery 方法，最后介绍与 Ajax 相关的一组 jQuery 工具函数。

12.3 jQuery 选择器

与 CSS 选择器一样，jQuery 选择器也用于从 HTML 文档中选择和匹配相关的元素（确切地说，是从 DOM 文档树中选择和匹配相关的 DOM 元素）。当然，它们从属于不同的语言和技术，并没有直接的关系。jQuery 参考并借鉴了 CSS 选择器的语法，并引入了大量的过滤选择器。下面介绍常用的 jQuery 选择器。

12.3.1 基本选择器

jQuery 的基本选择器对应于 CSS 的基本选择器，即 CSS 中的基本选择器在 jQuery 中同样可用。表 12-1 列出了基本选择器的语法、用例及其匹配说明。

表 12-1 基本选择器

选择器	用 例	匹 配
*	$("*")	所有元素
#<id 值>	$("#lastname")	id 属性值为"lastname"的元素
.<类名>	$(".intro")	所有 class 属性值为"intro"的元素
<元素名>	$("p")	所有<p>元素

续表

选择器	用　　例	匹　　配
[<属性名>]	$("[href]")	所有带有 href 属性的元素
[<属性名>=<属性值>]	$("[href='#']")	所有 href 属性的值等于"#"的元素
< selector1>,< selector2>,…	$("th,td")	所有 th 元素和 td 元素

12.3.2　层次选择器

jQuery 的层次选择器对应于 CSS 的层次选择器，即 CSS 中的层次选择器在 jQuery 中同样可用。表 12-2 列出了层次选择器的语法、用例及其匹配说明。

表 12-2　层次选择器

选择器	用　　例	匹　　配
<selector1> <selector2>	$("table tr")	所有<table>元素中的所有<tr>后代元素
<selector1> > <selector2>	$("ul > li")	所有 元素中的所有 子元素
< selector1> + < selector2>	$(".i1 + p")	id 属性值为"i1"的元素后面相邻的<p>兄弟元素

12.3.3　过滤选择器

过滤选择器通常用于对已预匹配的候选元素再做进一步过滤，找出最终的选中元素。过滤选择器以冒号（:）开头。这里介绍简单过滤选择器、内容过滤选择器和子元素过滤选择器。

1. 简单过滤选择器

简单过滤选择器的种类较多，大多数简单过滤选择器会以元素在所有候选元素中的位置来确定其是否匹配。这里各候选元素的索引号从 0 开始计数。表 12-3 列出了简单过滤选择器的语法、用例及其匹配说明。

表 12-3　简单过滤选择器

选择器	用　　例	匹　　配
:first	$("p:first")	第一个<p>元素
:last	$("p:last")	最后一个<p>元素
:even	$("tr:even")	所有偶数位置上的<tr>元素，索引号从 0 开始
:odd	$("tr:odd")	所有奇数位置上的<tr>元素，索引号从 0 开始
:eq(index)	$("ul li:eq(3)")	ul 元素中的索引号为 3 的 li 元素，index 从 0 开始
:gt(index)	$("ul li:gt(3)")	ul 元素中索引号大于 3 的 li 元素，索引号从 0 开始
:lt(index)	$("ul li:lt(3)")	ul 元素中索引号小于 3 的 li 元素，索引号从 0 开始
:not(<selector>)	$("p:not(.c1)")	所有 class 属性值不为"c1"的<p>元素
:header	$("div :header")	所有<div>元素中的标题元素（<h1>~<h6>）

2. 内容过滤选择器

内容过滤选择器主要以某元素的内容是否满足指定要求来确定其是否匹配。一个元素

的内容可以包含文本和子元素，每个子元素中又可以包含文本和子元素。表 12-4 列出了内容过滤选择器的语法、用例及其匹配说明。

表 12-4　内容过滤选择器

选择器	实　例	匹　配
:contains(<text>)	$("p:contains('PHP')")	内容包含文本"PHP"的所有<p>元素
:empty	$("p:empty")	不包含子元素和文本内容的所有<p>元素
:parent	$("div:parent")	所有包含子元素或文本内容的<div>元素
:has(<selector>)	$("div:has(p)")	所有包含<p>元素的<div>元素

3. 子元素过滤选择器

子元素过滤选择器把要选择的元素看作是某元素的子元素，主要以该子元素在其兄弟元素中的位置来确定其是否匹配。在这里，所有兄弟元素的索引号从 1 开始计数。表 12-5 列出了子元素过滤选择器的语法、用例及其匹配说明。

表 12-5　子元素过滤选择器

<table>
<tr><th>选择器</th><th>实　例</th><th>匹　配</th></tr>
<tr><td>:first-child</td><td>$("li:first-child")</td><td>所有在其兄弟元素中排序第 1 的<li>元素</td></tr>
<tr><td>:first-of-type</td><td>$(".c1:first-of-type")</td><td>所有在其同类型的兄弟元素中排序第 1 且 class 属性为"c1"的元素</td></tr>
<tr><td>:last-child</td><td>$("li:last-child")</td><td>所有在其兄弟元素中排序最后一个的<li>元素</td></tr>
<tr><td>:last-of-type</td><td>$(".c1:last-of-type")</td><td>所有在其同类型的兄弟元素中排序最后一个且 class 属性为"c1"的元素</td></tr>
<tr><td>:nth-child(even|odd|index|equation)</td><td>$("div p:nth-child(odd)")</td><td>所有在其兄弟元素中的排序位置为奇数，且属于<div>元素后代的<p>元素</td></tr>
<tr><td>:nth-of-type(even|odd|index|equation)</td><td>$("div p:nth-of-type(odd)")</td><td>所有在其同类型的兄弟元素中的排序位置为奇数，且属于<div>元素后代的<p>元素</td></tr>
<tr><td>:only-child</td><td>$("p:only-child")</td><td>所有没有任何兄弟元素的<p>元素</td></tr>
<tr><td>:only-of-type</td><td>$(".c1:only-of-type")</td><td>所有没有任何同类型兄弟元素，且其 class 属性为"c1"的元素</td></tr>
</table>

【例 12-4】 使用子元素过滤选择器。代码如下：

```
<!DOCTYPE html>
<html>
   <head>
      <title>12-4</title>
      <meta charset="UTF-8">
      <script src="jquery-3.1.0.min.js"></script>
   </head>
   <body>
      <ul>
```

```
            <li>item1</li><li>item2</li><li>item3</li>
        </ul>
        <ol>
            <li>项目 1</li><li>项目 2</li><li>项目 3</li><li>项目 4</li>
        </ol>
        <script>
            $("li:nth-child(even)").css( "background-color", "gray");
        </script>
    </body>
</html>
```

页面呈现时，所有在其兄弟元素中的排序位置为偶数的<li>元素都会以灰色背景颜色显示，包括“item2”“项目 2”和“项目 4”。

12.4　jQuery 操作 HTML 元素

这里介绍一组 jQuery 方法，它们可以对 HTML 元素的属性、样式、内容等进行操作。

12.4.1　操作元素属性

使用 attr 方法可以获取或设置元素属性，使用 removeAttr 方法可以删除元素属性。

（1）attr(<name>)。获取选中元素中第 1 个元素（索引值为 0）的指定属性的值。

（2）attr(<name>, <value>)。为选中元素的指定属性设置相应的值。

（3）attr({<name1>:<value1>, <name2>:<value2>})。提供一个对象，为选中元素的多个属性设置相应的值。

（4）attr(<name>, function(<index>, <value>))。用参数函数的返回值为选中元素的指定属性设置值，各选中元素会分别调用该函数一次。

参数函数的第 1 个参数 index 为当前元素在 jQuery 对象中的索引值，第 2 个参数 value 为当前元素在指定属性上原来的值。如果不需要这两个值，函数可以不带参数。

在参数函数体中，可以使用关键字 this 表示当前元素对象。如果需要在当前元素上调用 jQuery 方法，可先使用$(this)将其封装成一个 jQuery 对象。

指定参数函数既可以采用普通函数的方式，也可以采用匿名函数的方式。所谓普通函数的方式，就是先定义一个函数，然后将参数设置为函数名。所谓匿名函数方式，就是直接将参数设置为一个匿名函数。

（5）removeAttr(<name>)。从选中元素中删除指定的属性。

12.4.2　获取和设置表单值

通常，每个表单控件元素有个值。调用 val 方法可以获取或设置表单控件元素的值。

（1）val()。获取选中元素中第 1 个元素（应该是表单控件元素）的当前值。如果第 1 个元素是一个多值表单控件（设置了 multiple 属性的 select 元素），方法返回一个数组。

（2）val(<value>)。将选中元素中所有表单控件元素设置为指定的值。

（3）val(function(<index>, <value>))。用参数函数的返回值为选中元素中所有表单控件元素设置值，各表单控件元素会分别调用该函数一次。

如果要为检测框组、单选按钮组或选择列表设置值，应该指定一个数组，对某单选按钮组或单选的选择列表来说，指定的数组应该只包含一个值。当某个选项（类型为 radio 或 checkbox 的 input 元素，包含于 select 元素中 option 元素）的值与数组中的某个值相等时，该选项就被选中。

12.4.3　设置元素的样式

使用 css 方法可以设置元素的样式。

（1）css(<name>, <value>)。为选中元素的 style 属性添加指定的 CSS 声明，其中 name 为声明的属性名，value 为相应的属性值。

（2）css({<name1>:<value1>, <name2>:<value2>})。提供一个对象，为选中元素的 style 属性添加多个 CSS 声明。

（3）css(<name>, function(<index>, <value>))。为选中元素的 style 属性添加指定的 CSS 声明，其中 name 为声明的属性名，属性值为函数的返回值，各选中元素会分别调用函数一次。

12.4.4　设置元素的样式类

使用 addClass 和 removeClass 方法可以分别添加和删除元素的样式类，使用 toggleClass 方法可以在添加和删除样式类之间进行切换。

（1）addClass(<className>)。为选中元素的 class 属性添加一个或多个 CSS 样式类，两个样式类名之间用空格分隔。

（2）addClass(function(<index>, <className>))。为选中元素的 class 属性添加一个或多个 CSS 样式类，要添加的样式类名由参数函数返回，各选中元素会分别调用该函数一次。

（3）removeClass()。删除选中元素的 class 属性所包含的所有样式类。

（4）removeClass(<className>)。从选中元素的 class 属性中删除指定的一个或多个样式类。

（5）removeClass(function(<index>, <className>))。从选中元素的 class 属性中删除指定的一个或多个样式类，要删除的样式类名由参数函数返回，各选中元素会分别调用该函数一次。

（6）toggleClass(<className>)。当选中元素的 class 属性不包含指定的样式类时，就给元素添加这些类；反之，就从元素中删除这些类。

（7）toggleClass(function(<index>, <className>))。该方法也是在添加和删除样式类之间进行切换，只是要切换的样式类名由参数函数返回，各选中元素会分别调用该函数一次。

12.4.5　获取和设置元素内容

使用 html 和 text 方法可以获取和设置元素内容，使用 append 和 prepend 方法可以分别在元素原有内容后面和前面插入内容。

（1）html()。获取选中元素中第 1 个元素（索引值为 0）的内容，包括其中的 HTML

标签。

（2）text()。获取选中元素中所有元素的文本内容（去除 HTML 标签），并将其连接起来产生一个结果字符串。依赖于不同的浏览器实现，产生的结果字符串中包含的空白符号、换行符也许会存在差异。

（3）html(<htmlString>)。将选中元素的内容用指定的 HTML 字符串替换。

（4）html(function(<index>, <oldHtml>))。将选中元素的内容用参数函数返回的 HTML 字符串替换，各选中元素会分别调用函数一次。如果函数指定了参数，那么第 2 个参数<oldHtml>保存着当前元素原有的内容（包括 HTML 标签）。

（5）text(<text>)。将选中元素的内容用指定的文本字符串替换。如果指定的字符串中包含 HTML 标签，那么其中的“<”和“>”等特殊符号将被自动转换成相应的 HTML 实体表示。

（6）text(function(<index>, <oldText>))。将选中元素的内容用参数函数返回的文本字符串替换，各选中元素会分别调用该函数一次。如果函数指定了参数，那么第 2 个参数<oldText>保存着当前元素原有的内容（去除了 HTML 标签）。

（7）append(<content>)。在选中元素的内容后面插入指定的内容，插入的内容可以包含 HTML 标签。

（8）append(function(<index>, <oldHtml>))。在选中元素的内容后面插入函数返回的内容，各选中元素会分别调用函数一次。

（9）prepend(<content>)。在选中元素的内容前面插入指定的内容，插入的内容可以包含 HTML 标签。

（10）prepend(function(<index>, <oldHtml>))。在选中元素的内容前面插入函数返回的内容，各选中元素会分别调用函数一次。

12.4.6 删除元素

调用 jQuery 对象的 remove 方法可以从 DOM 文档树中删除选中元素本身。该方法的语法格式如下：

```
remove([<selector>])
```

如果没有指定参数<selector>，方法删除所有的选中元素。如果指定参数<selector>，那么仅删除与指定选择器相匹配的选中元素。

【例 12-5】 操作元素内容。页面呈现时，3 个段落都会在内容前加上序号，并以粗体显示。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <title>12-5</title>
    <meta charset="UTF-8">
    <script src="jquery-3.1.0.min.js"></script>
```

```
    </head>
    <body>
      <p>apple</p>
       <p>banana</p>
       <p>orange</p>
       <script>
         $("p").html(function(index, oldHtml) {
            return "<b>"+(index+1)+"."+oldHtml+"</b>";
         });
       </script>
     </body>
 </html>
```

12.5 jQuery 事件处理

通常，各种 DOM 元素都可以触发某些类型的事件。可以在某个 DOM 元素上为某种类型事件注册相应的事件处理函数，那么一旦这个 DOM 元素触发了这种事件，已注册的事件处理函数就会被自动执行。

jQuery 对事件的类型、特性，以及注册事件处理函数的 API，都做了详细的规范和定义。

12.5.1 常用的 jQuery 事件

大多数事件是支持冒泡的，称为冒泡事件。所谓冒泡是指当事件在某个元素上触发后，该元素的父元素也会随即触发这个事件，接着它的父元素的父元素又会触发该事件，这个过程不断继续，直至 document 和 window 对象。在整个过程中，事件就像水泡一样不断向 DOM 树的树根冒。

有些事件是不支持冒泡的，称为非冒泡事件，包括 blur、focus、mouseenter 和 mouseleave 等。

下面是一些常用的 jQuery 事件。

（1）blur。当元素失去焦点时触发。在传统浏览器中，该事件仅适用于表单控件元素，在新型浏览器中，该事件适用于所有可聚焦的元素。该事件不支持冒泡。

（2）change。当元素的值发生改变时触发。该事件适用于 input、textarea 和 select 等元素。当用于单选按钮、复选框和 select 元素时，change 事件会在选择某个选项时发生。当用于其他输入域时，该事件会在元素失去焦点时发生。

（3）click。当单击元素（鼠标按钮被按下又释放）时触发，适用于所有的元素。

（4）contextmenu。当右击元素时（打开上下文菜单前）触发，适用于所有元素。

（5）dblclick。当双击元素时触发，适用于所有的元素。

（6）focus。当元素获得焦点时触发。在传统浏览器中，该事件仅适用于表单控件元素，在新型浏览器中，该事件适用于所有元素。通过设置元素的 tabindex 属性，可使元素成为可聚焦的。该事件不支持冒泡。

（7）keydown。当元素聚焦而用户在键盘上按下一个键时触发，适用于所有可聚焦的

元素。

（8）keypress。当用户在键盘上输入一个可打印字符时，在 keydown 和 keyup 两个事件之间会触发一个 keypress 事件。该事件适用于所有可聚焦的元素。

（9）keyup。当元素聚焦而用户在键盘上释放一个键时触发，适用于所有可聚焦的元素。

（10）load。当一个元素和其所有子元素都已经完全装入时触发。该事件适用于任何带有 URL 的元素，如<img>元素、window 对象。

（11）mouseenter。当鼠标指针进入元素时触发，适用于所有的元素。该事件不支持冒泡。

（12）mouseleave。当鼠标指针离开元素时触发，适用于所有的元素。该事件不支持冒泡。

（13）submit。当要提交表单时触发。该事件仅适用于<form>元素。

（14）unload。当用户离开当前页面时触发，适用于 window 对象。产生离开当前页面的情形包括：单击页面中的超链接；在地址栏中输入新的 URL；单击前进或后退按钮；单击刷新当前页面；关闭当前窗口。

12.5.2　注册和注销事件处理函数

注册事件处理函数就是在某元素上为某种事件绑定一个处理函数，一旦这种事件在该元素上发生，事先绑定的处理函数就会被自动执行。注销事件处理函数就是从某元素上移去已经注册的事件处理函数。

1. 用便捷方式注册事件处理函数

作为一种便捷方式，可以调用 jQuery 对象的与事件名同名的方法，在选中的 DOM 元素上为相应事件注册事件处理函数，其格式如下：

```
$(<selector>).<event_name>(<function_name>)
```

或

```
$(<selector>).<event_name>(function([<event>]) {...})
```

其中<event_name>指定事件名，也是方法名。第 1 种格式采用普通函数的方式指定事件处理函数；第 2 种格式采用匿名函数的方式指定事件处理函数。不管是哪种方式，事件处理函数都可以定义一个可选的形参<event>，用于接收当前事件对象。

例如，如下代码：

```
function f1() {
   alert("f1");
}
$("#div1").click(f1);
```

或

```
$("#div1").click(function() {
   alert("f1");
```

```
});
```

具有相同的功能，都是在 id 值为"div1"的元素上为"click"事件注册了一个处理函数，只是前者采用普通函数的方式，而后者采用了匿名函数的方式。

2. 用 on 方法注册事件处理函数

可以调用 jQuery 对象的 on 方法，在选中的 DOM 元素上为指定的一个或多个事件注册事件处理函数，其格式如下：

```
on(<event_names> [, <selector>] [, <data>], <function_name>)
```

或

```
on(<event_names> [, <selector>] [, <data>], function([<event>]) {...})
```

其中，<event_names>指定一个或多个事件名，即在多种事件上注册相同的处理函数，两个事件名之间用空格分隔；<function_name>指定事件处理函数的函数名，也可以采用匿名函数的方式。

如果省略<selector>或将其指定为 null，那么该方法就如同上述便捷方式一样，注册一个被称为直接的事件处理函数，用于处理在选中元素上触发的指定的事件，此事件可以直接触发于选中元素，也可以产生于其内部的某个后代元素，然后经过冒泡过程导致选中元素触发。

如果指定<selector>，该方法注册一个称为委托的事件处理函数，它不用于处理选中元素本身触发的事件，而是用于处理选中元素内部与<selector>匹配的后代元素触发的指定事件。

例如，如下代码：

```
function f2() {
  alert("f2");
}
$("#div1").on("click", "p", f2);
```

在 id 值为"div1"的元素上注册了一个委托处理函数，用于处理在其后代<p>元素上触发的"click"事件。

需要注意的是，在委托处理函数中，关键字 this 表示的是触发事件的后代元素，而不是事件处理函数注册的所在元素。

可以指定一个额外的数据<data>。如果指定，该数据会被封装在事件对象（<event>）中。在事件处理函数中，访问事件对象的 data 属性可以获得该数据。

3. 注销事件处理函数

要注销事件处理函数，可以调用 jQuery 对象的 off 方法，方法注销在选中元素上为指定事件注册的指定的事件处理函数。格式如下：

```
off()
```

或

```
off(<event_names> [, <selector> ] [, <function_name>])
```

如果不带任何参数，方法注销在选中元素上为所有事件注册的所有事件处理函数，不管是直接的处理函数，还是委托的处理函数。例如，如下代码：

```
$("p").off();
```

代码注销在段落元素上为各种事件注册的各种直接的或委托的处理函数。

如果仅指定参数<event_names>，方法注销在选中元素上为指定事件注册的所有事件处理函数，不管是直接的处理函数，还是委托的处理函数。例如，如下代码：

```
$("p").off("click");
```

用于注销在段落元素上为"click"事件注册的各种直接的或委托的处理函数。

如果指定参数<selector>，方法注销在选中元素上注册的、用于处理匹配的后代元素触发的指定事件的委托处理函数。其中，<selector>是选择器字符串，用于匹配选中元素中的某些后代元素，必须与 on 方法中指定的一致。

如果将<selector>设置为"**"，那么方法注销在选中元素上注册的、用于处理任意后代元素触发的指定事件的委托处理函数。例如，如下代码：

```
$("#div1").off("click", "**");
```

用于注销在 id 值为"div1"的元素上注册的、用于处理任意后代元素触发的"click"事件的委托处理函数。

如果指定参数<function_name>，方法注销指定的处理函数。例如，如下代码：

```
$("#div1").off("click", f1);
```

用于注销在 id 值为"div1"的元素上为"click"事件注册的处理函数 f1（直接或委托的）。

又例如，如下代码：

```
$("#div1").off("click", "p", f2);
```

用于注销在 id 值为"div1"的元素上注册的、用于处理后代段落元素触发的"click"事件的委托处理函数 f2。

12.5.3 事件对象

在定义事件处理函数时，可以指定一个形参，该参数会接收一个 jQuery 事件对象。访问事件对象的属性，可以了解事件的一些具体细节，调用事件对象的方法，可以影响事件的某种行为。

下面是事件对象的一些属性。

（1）type。当前事件对象的事件类型名，例如"click" "load" "submit"等。

（2）currentTarget。当前 DOM 元素，与 this 表示同一个元素。对于直接事件处理函数，表示处理函数注册所在的元素；对于委托事件处理函数，表示触发事件的匹配后代元素。

（3）target。最初触发该事件的 DOM 元素。

（4）screenX、screenY。适用于鼠标事件，返回鼠标指针相对于显示器左上角的 X 坐标和 Y 坐标。

（5）clientX、clientY。适用于鼠标事件，返回鼠标指针相对于所在窗口左上角的 X 坐标和 Y 坐标。

（6）pageX、pageY。适用于鼠标事件，返回鼠标指针相对于当前文档左上角的 X 坐标和 Y 坐标。

（7）which。适用于键盘或鼠标事件，指定哪个键或按钮被按下。对鼠标事件来说，1 表示左键，2 表示中键，3 表示右键。对键盘事件来说，返回可打印字符的编码。

（8）data。在用 on 方法注册一个事件处理函数时，可以有选择地提供一个参数<data>。这个参数数据就保存在该属性中。

（9）result。事件处理函数可以有返回值，并保存在该属性中。在一个事件冒泡过程中，某个事件处理函数可以通过访问该属性，获得上一个事件处理函数的返回值。

下面是两个常用的事件对象的方法。

（1）preventDefault()。告诉浏览器不要执行与当前事件关联的默认行为。有些事件会有相关联的默认行为，如单击（click）一个超链接将导航到目标页面，提交（submit）表单将收集表单数据并产生一个 HTTP 请求等。

（2）stopPropagation()。停止当前事件继续冒泡，防止前辈元素的事件处理函数被通告和执行。

如果事件处理函数返回一个 false 值，那么它既会阻止与当前事件关联的默认行为的执行，也会防止当前事件进一步冒泡。

【例 12-6】 验证表单数据，如果文本域为空，则显示错误信息并阻止表单提交。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <title>12-6</title>
    <meta charset="UTF-8">
    <script src="jquery-3.1.0.min.js"></script>
    <script>
      $(function(){
        $("#f1").submit(function(event){
          var tfs= $("#f1 [type=text]"); // 所有的文本域
          var flag = true;
          for(n=0; n<tfs.length; n++) { // 遍历所有文本域
            if(tfs.eq(n).val()==="") { // 当前文本域为空
              tfs.eq(n).next().text("是必填项，不能为空");
              flag = false;
            } else {
              tfs.eq(n).next().text("");
            }
```

```
            }
            if(!flag) {               // 如果有文本域为空
              event.preventDefault(); // 阻止表单提交
            }
        });
      });
    </script>
  </head>
  <body>
    <form id="f1" method="post" action="process.php">
      <p>
        <input type="text" name="a" value="" />
        <span style="color: red"></span>
      </p>
      <p>
        <input type="submit" value="OK" />
      </p>
    </form>
  </body>
</html>
```

12.6 jQuery 动画效果

jQuery 提供了一套支持各种动画效果的方法。这里介绍 3 组方法，它们能够以各自的动画效果显示和隐藏选中元素。控制元素显示或隐藏的最基本的 CSS 属性是 display，当该属性被设置为 none 时，元素被隐藏；当被设置为其他值时，元素被显示。要使元素的显示和隐藏具有动画效果，还会用到其他 CSS 属性。

注意：如果选中元素有多个，那么这些元素的隐藏和显示过程是同时进行的，而不是一个接一个进行的。

12.6.1 淡出与淡入

通过连续地降低或增加元素的不透明度（CSS 的 opacity 属性），使元素产生淡出或淡入的动画效果。

1. 淡出

可以调用 fadeOut 方法以淡出的动画效果隐藏选中的元素，其格式如下：

```
fadeOut([<duration>] [, <function_name>])
```

或

```
fadeOut([<duration>] [, function() {...}])
```

淡出的过程是指元素的透明度由完全不透明到完全透明(opacity 属性值由 1 渐变为 0)，

最终隐藏元素（display 属性设置为 none）。

参数<duration>指定淡出过程持续的时间，以毫秒为单位，默认值为 400。持续时间越短，淡出隐藏的速度越快。如果将该参数设置为 0，则元素直接隐藏，没有动画效果。

如果指定参数函数<function_name>，那么在选中元素隐藏完成后，各元素都会分别调用该函数一次。

2. 淡入

可以调用 fadeIn 方法以淡入的动画效果显示选中的元素，其格式如下：

```
fadeIn([<duration>] [, <function_name>])
```

或

```
fadeIn([<duration>] [, function() {...}])
```

淡入的过程是指显示元素（display 属性设置为非 none），同时元素的透明度由完全透明到完全不透明（opacity 属性值由 0 渐变为 1）。

参数<duration>和<function_name>的作用与淡出方法 fadeOut 中的类似。

3. 淡出或淡入

可以调用 fadeToggle 方法以淡出或淡入的动画效果隐藏或显示选中的元素，其格式如下：

```
fadeToggle([<duration>] [, <function_name>])
```

或

```
fadeToggle([<duration>] [, function() {...}])
```

若选中元素原来是隐藏的，就以淡入方式显示它；若选中元素原来是显示的，就以淡出方式隐藏它。参数<duration>和<function_name>的作用与 fadeOut 和 fadeIn 方法中的类似。

12.6.2 滑动

通过连续地改变元素的纵向尺寸（CSS height、padding-top、padding-bottom、margin-top 和 margin-bottom 等属性），使元素产生滑动的动画效果。

1. 以滑动方式隐藏元素

可以调用 slideUp 方法以滑动的动画效果隐藏选中的元素，其格式如下：

```
slideUp([<duration>] [, <function_name>])
```

或

```
slideUp([<duration>] [, function() {...}])
```

滑动隐藏的过程是指元素的纵向尺寸由正常逐渐变小为 0，并最终隐藏元素（display 属性设置为 none）。

其中参数<duration>指定滑动隐藏的持续的时间，以毫秒为单位，默认值为 400。持续

时间越短，滑动隐藏的速度越快。如果将该参数设置为 0，则元素直接隐藏，没有动画效果。

如果指定参数函数<function_name>，那么在选中元素隐藏完成后，各元素都会分别调用该函数一次。

2. 以滑动方式显示元素

可以调用 slideDown 方法以滑动的动画效果显示选中的元素，其格式如下：

```
slideDown([<duration>] [, <function_name>])
```

或

```
slideDown([<duration>] [, function() {...}])
```

滑动显示的过程是指显示元素（display 属性设置为非 none），同时元素的纵向尺寸由 0 逐渐变大为正常。

参数<duration>和<function_name>的作用与滑动隐藏方法 slideUp 中的类似。

3. 以滑动方式隐藏或显示元素

可以调用 slideToggle 方法以滑动的动画效果隐藏或显示选中的元素，其格式如下：

```
slideToggle([<duration>] [, <function_name>])
```

或

```
slideToggle([<duration>] [, function() {...}])
```

若选中元素原来是隐藏的，就以滑动方式显示它；若选中元素原来是显示的，就以滑动方式隐藏它。参数<duration>和<function_name>的作用与 slideUp 和 slideDown 方法中的类似。

12.6.3 显示与隐藏

这组方法的名字（hide、show 和 toggle）不含动画的含义，但实际上它们也可以产生动画效果，而且其效果结合了滑动和淡出淡入两方面的因素。一方面，它会通过逐渐改变元素的尺寸（CSS width、height、padding 和 margin）来获得动画效果。与滑动效果相比，它不仅改变纵向尺寸，也改变横向尺寸。另一方面，它也会通过改变元素的透明度来产生淡出淡入的效果。

1. 隐藏元素

可以调用 hide 方法隐藏选中的元素，其格式如下：

```
hide([<duration>] [, <function_name>])
```

或

```
hide([<duration>] [, function() {...}])
```

参数<duration>指定隐藏的持续的时间，以毫秒为单位。与淡入淡出和滑动不同，在这里，该参数的默认值为 0。所以如果不指定参数<duration>，选中的元素立即隐藏，没有动

画效果。

如果将参数<duration>指定为非零，那么方法将以动画方式隐藏选中元素：各元素的各种尺寸由正常逐渐变小为 0，元素的透明度由完全不透明逐渐变为完全透明，最终元素被隐藏（display 属性设置为 none）。

如果指定参数函数<function_name>，那么在选中元素隐藏完成后，各元素都会分别调用该函数一次。

2. 显示元素

可以调用 show 方法显示选中的元素，其格式如下：

```
show([<duration>] [, <function_name>])
```

或

```
show([<duration>] [, function() {...}])
```

其中，参数<duration>和<function_name>的作用与 hide 方法中的类似。

当将参数<duration>指定为非零时，方法将以动画方式显示选中元素：各元素被显示（display 属性被设置为非 none），同时各元素的各种尺寸由 0 逐渐变大为正常，各元素的透明度由完全透明逐渐变为完全不透明。

3. 隐藏或显示元素

可以调用 toggle 方法隐藏或显示选中的元素，其格式如下：

```
toggle([<duration>] [, <function_name>])
```

或

```
toggle([<duration>] [, function() {...}])
```

若选中元素原来是隐藏的，就显示或以动画效果显示它；若选中元素原来是显示的，就隐藏或以动画效果隐藏它。参数<duration>和<function_name>的作用与 hide 和 show 方法中的类似。

【例 12-7】 通过单击按钮，切换所有段落元素的显示和隐藏。代码如下：

```
<!DOCTYPE html>
<html>
  <head>
     <title>12-7</title>
     <meta charset="UTF-8">
     <script src="jquery-3.1.0.min.js"></script>
     <script>
     $(function(){
        $("input[type=button]").click(function() {
            $("p").toggle(200);
         });
      });
```

```
        </script>
    </head>
    <body>
        <input type="button" value="Toggle" />
        <p>Hello</p>
        <p style="display: none">Good Bye</p>
    </body>
</html>
```

页面上除了一个按钮，还有两个段落。一开始，第 1 个段落是显示的，第 2 个段落是隐藏的。当单击按钮时，两个段落分别改变它们的显示和隐藏的状态：原来是显示的，变为隐藏；原来是隐藏的，变为显示。

12.7 jQuery 中的 Ajax

jQuery 提供多个与 Ajax 有关的函数和方法。这些函数和方法，隐藏了 Ajax 方法的一些细节以及不同浏览器之间的差异，可以更方便地向远程服务器请求文本、HTML、XML 和 JSON 等数据，并且能够很容易地把这些外部数据载入网页的相关元素中。

12.7.1 get 和 post 函数

get 和 post 都是 jQuery 中的工具函数，调用时用 jQuery 或$作为前缀，例如$.get(…)，用于向服务器发送异步的 HTTP 请求，以获取响应数据。这里，get 函数发送 GET 请求，post 函数发送 POST 请求。两个函数的语法格式如下：

```
get(<url> [, <data>] [, <function_name>] [, <dataType>])
post(<url> [, <data>] [, <function_name>] [, <dataType>])
```

或

```
get(<url> [, <data>] [, function(responseData) {...}] [, <dataType>])
post(<url> [, <data>] [, function(responseData) {...}] [, <dataType>])
```

其中：

（1）<url>：指定被请求的资源的 URL。

（2）<data>：是一个可选项，用于指定请求参数，可以是一个字符串或简单对象。例如，下面两种格式都指定 name 和 time 两个请求参数，前者是字符串，后者是简单对象。

```
"name=John&time=2pm"
{"name": "John", "time": "2pm"}
```

（3）<function_name>：一个可选的回调函数。该回调函数可以指定一个参数，用以接收请求返回的响应数据。响应数据可能是响应体内容，也可能是由响应体内容转换而来的 DOM 文档树、JSON 对象等。

该回调函数只在请求成功完成且数据转换正确的情况下被调用。函数体可以对响应数据做所需的处理。

（4）<dataType>：是一个可选项，用于指定响应数据的类型，即要传递给回调函数的数据的类型，是一个字符串，例如："xml" "json" "text" "html"等。如果没有设置该参数，函数通常会根据响应头中 Content-Type 域的值自动确定响应数据的类型。例如，如果请求的是一个 JSON 文件，那么默认情况下响应数据将是一个 JSON 对象。

【例 12-8】 用 jQuery 中的 Ajax 工具函数实现例 12-1 中的“录入提示”功能。这里，取消了 clienthint.js 文件，实现类似功能的 jQuery 代码直接放置在页面文件 12-8.html 中。该页面文件的代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <title>12-8</title>
    <meta charset="UTF-8">
    <script src="jquery-3.1.0.min.js"></script>
    <script>
      $(function() {
        $("#txt1").keyup(function() {
          var pre = $(this).val();
          if (pre.length === 0) {
            $("#txtHint").html("");
            return;
          }
          $.get("gethint.php", {pre:pre}, function(responseData) {
            $("#txtHint").html(responseData);
          });
        });
      });
    </script>
  </head>
  <body>
    <form>
      输入单词:
      <input type="text" id="txt1" />
    </form>
    <p>提示: <span id="txtHint"></span></p>
  </body>
</html>
```

服务器端的 gethint.php 文件的代码不变。

12.7.2 请求 JSON 数据

这里，首先介绍 JSON 数据的概念、格式，然后介绍专门用于从远程服务器获取和处理 JSON 数据的 jQuery 工具函数 each 和 getJSON。

1. JSON

JSON（JavaScript Object Notation，JavaScript 对象表示法）是一种数据对象表示法，用于在各种语言之间存储和交换文本信息。JSON 的作用类似 XML，但比 XML 更小、更快，更容易解析。

JSON 使用 JavaScript 中表示数组和对象的语法，即使用方括号[]表示数组，用花括号（{ }）表示对象。

例如，下面表示一个包含 3 个元素的 JSON 数组：

```
["item1", "item2", "item3"]
```

每个数组元素可以是数值、字符串、布尔值，也可以是对象或数组本身。字符串要用双引号作为定界符。

又例如，下面表示一个包含两个属性的 JSON 对象。

```
{"name1": "value1", "name2": "value2"}
```

对象中每个属性的名字必须是字符串，属性值可以是数值、字符串、布尔值，也可以是数组或对象本身。字符串要用双引号作为定界符。

通常，JSON 数据会以文本字符串的形式在应用之间或在网络上传递。当一个应用接收到 JSON 数据后，可以用相应的工具程序对其进行解析、进而把它还原成对象或数组。

2. each 函数

each 作为 jQuery 的一个工具函数，调用时用 jQuery 或$作为前缀，如$.each(…)，用于迭代处理对象中的每个属性，或数组中的每个元素。其语法格式如下：

```
each(<object>, <function_name>)
each(<array>, <function_name>)
```

或

```
each(<object>, function(<name>, <value>) {...})
each(<array>, function(<index>, <value>) {...})
```

第 1 个参数是对象或数组，第 2 个参数指定一个函数。如果第 1 个参数是对象，那么会为对象的每个属性调用一次指定的函数，传入属性名和属性值两个参数。如果第 1 个参数是数组，那么会为数组的每个元素调用一次指定的函数，传入元素的索引和元素值两个参数。如果指定的函数在处理某个属性或元素时返回 false，则迭代处理停止。

3. getJSON 函数

getJSON 是 jQuery 中的工具函数，调用时用 jQuery 或$作为前缀，如$.getJSON(…)，用于向服务器发送异步的 HTTP GET 请求，以获取 JSON 数据。其语法格式如下：

```
getJSON(<url> [, <data>] [, <function_name>])
```

或

```
getJSON(<url> [, <data>] [, function(<jsonData>) {...}])
```

其中参数如下。

（1）<url>：指定被请求的资源的 URL。该资源产生的响应体内容应该是 JSON 对象格式或数组格式的文本字符串。当方法接收到该内容后，将自动转换为 JSON 对象或数组，这里将它们统称为 JSON 数据。

（2）<data>：是一个可选项，用于指定请求参数，可以是一个字符串或简单对象。

（3）<function_name>：一个可选的回调函数。该回调函数可以指定一个参数，用以接收请求返回的 JSON 数据。

该回调函数只在请求成功完成且 JSON 数据正确生成的情况下被调用。函数体可以对 JSON 数据做进一步的处理。

【例 12-9】 当页面呈现时，从服务器获得一个 JSON 数组，然后将各数组元素编制为一个个的 HTML 列表项元素，最后把它们插入到页面的列表中。

该例包含两个文件，一个是页面文件 12-9.html，另一个是提供 JSON 数组的 PHP 文件 jsonforlist.php。当请求 12-9.html 时，浏览器窗口显示 item1、item2、item3、item4 和 item5 这 5 个列表项。

下面是页面文件 12-9.html 的代码：

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.      <title>12-9</title>
5.      <meta charset="UTF-8">
6.      <script src="jquery-3.1.0.min.js"></script>
7.      <script>
8.         $(function() {
9.            $.getJSON("jsonforlist.php", function(responseData) {
10.               htmlstr = "";
11.               $.each(responseData, function(index, value) {
12.                  htmlstr = htmlstr + "<li>"+value+"</li>";
13.               });
14.               $("ul").html(htmlstr);
15.            });
16.         });
17.      </script>
18.   </head>
19.   <body>
20.      <ul>
21.      </ul>
22.   </body>
23. </html>
```

上述第 9 行代码异步请求 PHP 文件 jsonforlist.php，获取一个 JSON 数组，下面是该文件的代码：

```
<?php
$data = '["item1"';
for($i=2; $i<=5; $i++) {
   $data = $data.', "item'.$i.'"';
}
echo $data."]";
?>
```

12.7.3 load 方法

load 是一个 jQuery 方法，通过 jQuery 对象调用，用于从服务器加载数据。首先它会自动向服务器发送异步的 HTTP 请求；然后接收服务器的响应，获取响应体内容，响应体内容简称响应内容，应该是一个 HTML 字符串；最后用响应内容替换所有选中元素的内容。如果当前 jQuery 对象不含任何选中元素，load 方法不执行任何操作。

load 方法的语法格式如下：

```
load(<url> [, <data>] [, <function_name>])
```

或

```
load(<url>[, <data>] [,function(<responseText>,<textStatus>, <xhr>) {...}])
```

其中参数如下。

（1）<url>：指定被请求的资源的 URL。如果请求成功，就用响应内容替换所有选中元素的内容；如果请求不成功，则不做替换操作。

<url>中也可以包含一个空格，此时空格后面的内容被看作是一个 jQuery 选择器。这种情况下，不把响应内容作为整体去替换，而只把响应内容中与该选择器相匹配的元素取出，并用它去替换选中元素的内容。

（2）<data>：是一个可选项，用于指定请求参数，可以是一个字符串或简单对象。如果提供请求参数且为对象形式，那么发送 POST 请求；否则发送 GET 请求。

（3）<function_name>：可以可选地指定一个回调函数。如果指定，则无论本次请求是否成功，该函数总会被调用。该函数在所有选中元素内容被替换后被调用，每个选中元素调用一次。

该回调函数可以有选择地指定参数用于接收相关信息。<responseText>是响应（体）内容，通常应该是一个 HTML 字符串。<textStatus>是 jQuery 中的 Ajax 响应状态文本，例如，"success"表示请求成功完成，"error"表示请求没有成功完成。<xhr>是 load 方法执行时发送 HTTP 请求所用的 XHR 对象。

【例 12-10】 页面一开始显示一个书名（用段落元素呈现）。单击该书名会从服务器装载并显示该图书的相关信息。

该例包含两个文件，一个是页面文件 12-10.html，另一个是提供图书信息（HTML 字符串）的 PHP 文件 htmlforload.php。

下面是页面文件 12-10.html 的代码：

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>12-10</title>
5.     <meta charset="UTF-8">
6.     <script src="jquery-3.1.0.min.js"></script>
7.     <script>
8.       $(function() {
9.         $("p").click(function() {
10.           id= $(this).attr("id");
11.           $(this).next("div").load("htmlforload.php", {"id": id});
12.         });
13.       });
14.     </script>
15.   </head>
16.   <body>
17.     <div>
18.       <p id="123" style="cursor: pointer">Web 应用开发</p>
19.       <div></div>
20.     </div>
21.   </body>
22. </html>
```

下面是 PHP 文件 htmlforload.php 的代码：

```
<?php
$id = $_POST["id"];
if($id==="123") {
  echo "<p>作者：张三</p>";
  echo "<p>出版社：清华大学出版社</p>";
  echo "<p>单价：32.00</p>";
}
```

这里，第 11 行代码调用 load 方法，由于以对象形式提供请求参数 id，所以客户端发送异步的 HTTP POST 请求。相应地，服务器端的 htmlforload.php 文件应该访问$_POST 数组以获取该请求参数，即$_POST["id"]）。

习　题　12

1．根据要求写 jQuery 代码

（1）创建 jQuery 对象，包含 id 值为"id1"的元素的第一个段落子元素。

（2）创建 jQuery 对象，包含<tbody>元素中所有奇数位置上的<tr>子元素。

（3）假设 jqo 是一个 jQuery 对象。创建 jQuery 对象，包含 jqo 对象中每个元素的后面一个兄弟元素。

（4）假设 jqo 是一个 jQuery 对象。请为 jqo 中的所有选中元素设置一个样式：CSS 属性名为"background-color"，属性值为"gray"。

（5）假设 jqo 是一个 jQuery 对象。请为 jqo 中的所有选中元素添加或删除样式类 C1：如果选中元素的 class 属性原先不包含样式类 C1，就给元素添加该类；反之，就从元素中删除该类。

（6）假设 jqo 是一个 jQuery 对象。请将 jqo 中的所有选中元素的内容清空。

（7）假设 jqo 是一个 jQuery 对象，其中包含一些文本域或口令域等选中元素。请在这些选中元素上为 blur 事件注册一个处理函数，该事件处理函数的功能只是用 alert 函数显示一下当前选中元素的当前取值。

（8）假设页面中有如下表单元素：

```
<form id="f1" action="process.php">
  <p><input id="i1" name="i1" type="text" value="" /></p>
  <p><input id="i2" type="submit" value="OK" /></p>
</form>
```

请在表单元素上为 submit 事件注册一个处理函数，该事件处理函数的功能是：如果文本域的值小于 0，就阻止表单提交。

（9）假设 jqo 是一个 jQuery 对象。请在 jqo 中的选中元素上为 click 事件注册一个处理函数，该事件处理函数的功能是，以滑动方式隐藏或显示 id 值分别为"id1"和"id2"的两个元素，如果元素原先是显示的，就隐藏它；否则就显示它。

（10）利用 jQuery 工具函数向服务器发送一个 GET 请求：要请求的资源的 URL 是"p.php"；要传送的请求参数的参数名为"x"、参数值为"abc"；若请求成功，返回时调用一个匿名函数，函数的功能是用 alert 函数显示响应体的内容。

2. 简答题

（1）什么是 Ajax？其最主要的特点是什么？

（2）什么是 jQuery？jQuery 涉及哪些功能？

（3）什么 jQuery 对象？举例说明创建 jQuery 对象的方法。

附录 A　上 机 实 验

教材正文中各实例介绍了教务选课系统管理员子系统的开发。本上机实验要求完成教务选课系统学生-教师子系统的开发。

各实验创建的文件保存于以下 3 个位置：

（1）教务选课系统项目 xk 中的“源文件”结点。

（2）教务选课系统项目 xk 中的“源文件”结点下的 student 子目录。

（3）教务选课系统项目 xk 中的“源文件”结点下的 teacher 子目录。

各实验访问的数据表都属于数据库 elective_manage。

各实验都可以访问“源文件”结点下 css 子目录、image 子目录和 lib 子目录中的各种文件。

A.1　实验 1：页面头和页面脚

本实验创建页面头模块文件 header.php 和页面脚模块文件 footer.php。这两个模块文件都保存于教务选课系统项目 xk 中的“源文件”结点。

A.1.1　目的与要求

（1）熟悉 PHP 运行环境 XAMPP，掌握基本的操作方法。

（2）熟悉 PHP 开发环境 NetBeans IDE，掌握常用的操作方法。

（3）熟悉和掌握 HTML 标签的使用。

（4）熟悉和掌握 CSS 样式与样式表的定义和应用。

A.1.2　实验内容

（1）创建 PHP 文件 header.php，作为能够呈现页面头的模块文件。该页面头的左边是系统的 logo 图标，中间是系统的欢迎信息，右边显示日期和时间。这里，日期和时间可以都先制作成静态的文本，在后面的实验中再加以改进。运行该模块文件的呈现效果如图 A-1 所示。



图 A-1　页面头

（2）创建 PHP 文件 footer.php，作为能够呈现页面脚的模块文件。除了一条水平线段，

该页面脚包括版权、单位名称、联系电话和电子邮件地址等信息。运行该模块文件的呈现效果如图 A-2 所示。



图 A-2　页面脚

A.2　实验 2：注册表单和登录表单

本实验初步创建用于学生注册的模块文件 registration.php 和用于登录的模块文件 login.php。这两个模块文件都保存于教务选课系统项目 xk 中的“源文件”结点。

本实验主要完成注册表单和登录表单外观的设计，实验 7 将对这两个模块文件进行修改，实现相关的注册和登录功能。

A.2.1　目的与要求

（1）理解表单的概念和作用。

（2）掌握表单元素和各种表单控件元素的使用，掌握表单的设计。

（3）掌握有关 CSS 属性的应用。

（4）掌握利用 CSS 技术实现页面布局的技术。

A.2.2　实验内容

（1）创建 PHP 文件 registration.php，作为用于学生注册的模块文件。本实验仅设计表单的外观，其注册功能将在实验 7 实现。运行该模块文件的呈现效果如图 A-3 所示。

该表单用于学生注册，具体要求如下：

① 用 CSS 技术（不用 HTML <table>标签）实现表单及表单控件的布局。

② 将表单标签的 method 属性值设置为"POST"。将 action 属性值设置为""，或缺省该属性，使表单提交时总是请求该模块文件（或其所在的页面文件）。

③ 除性别外，其他各输入域后面都有可能要显示错误信息，这些错误信息将用红色显示，请使用外部样式表 xk.css 中的合适样式。

（2）创建 PHP 文件 login.php，作为用于学生和教师登录的模块文件。本实验仅设计表单的外观，其登录功能将在实验 7 实现。运行该模块文件的呈现效果如图 A-4 所示。

该表单用于学生或教师的登录，具体要求如下：

① 用 CSS 技术（不用 HTML <table>标签）实现表单及表单控件的布局。

② 将表单标签的 method 属性值设置为"POST"。将 action 属性值设置为""，或缺省该属性，使表单提交时总是请求该模块文件（或其所在的页面文件）。

③ 用户名和密码两个输入域后面有可能要显示错误信息，这些错误信息将用红色显示，请使用外部样式表 xk.css 中的合适样式。



图 A-3　学生注册表单



图 A-4　学生、教师登录表单

A.3　实验 3：动态导航栏

本实验创建学生-教师子系统的 3 个动态导航栏模块文件：

（1）navigation.php：主导航栏模块文件，用于学生-教师子系统主页、注册页面和登录页面。保存于教务选课系统项目 xk 中的“源文件”结点。

（2）navigation_student.php：学生子系统导航栏模块文件，用于学生子系统各页面。保存于教务选课系统项目 xk 中的“源文件”结点下的 student 子目录。

（3）navigation_teacher.php：教师子系统导航栏模块文件，用于教师子系统各页面。保存于教务选课系统项目 xk 中的“源文件”结点下的 teacher 子目录。

A.3.1 目的与要求

（1）掌握 PHP 代码与 HTML 代码在 PHP 页面文件中交替使用的方法。
（2）掌握 PHP 变量的定义与使用。
（3）掌握 PHP 运算符、表达式的使用。
（4）掌握 PHP 流程控制语句的使用。

A.3.2 实验内容

（1）创建 PHP 文件 navigation.php，作为能够呈现主导航栏的模块文件，用于学生-教师子系统的主页、注册页面和登录页面。

该模块文件的执行要用到 4 个变量，当这些变量取不同值时会有不同的呈现效果。下面是这 4 个输入变量的变量名、含义及可能的取值：

```
$isLogon = {true|false}      // 是否处于登入状态，取 true 或 false
$lb = {同学|老师}             // 登录人员的类别，取"同学"或"老师"
$name = <姓名>                // 当前登录人员的姓名
$choice = {0|1|2}            // 用户选择状态，取 0、1 或 2
```

当处于未登录状态时（输入变量$isLogon 的值为 false），主导航栏的呈现效果如图 A-5 所示。

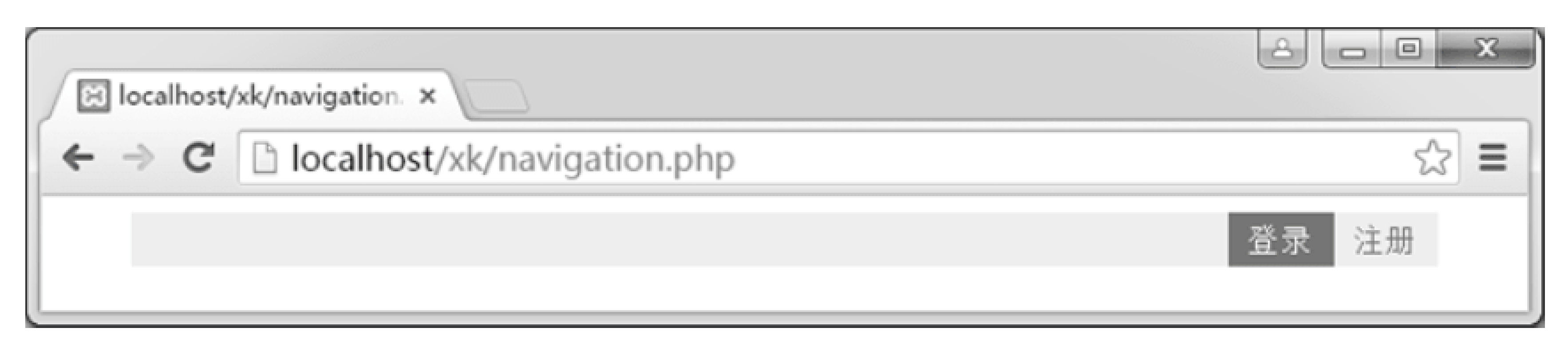


图 A-5　学生-教师子系统主导航栏（未登录状态）

其中，“登录”和“注册”是两个超链接的文本，两个超链接的 href 属性值可以分别设置为"login_p.php"和"registration_p.php"。两个超链接文本应该根据输入变量$choice 的取值，决定是否醒目显示。当$choice 为 1 时，“登录”文本醒目显示，当$choice 为 2 时，“注册”文本醒目显示，当$choice 为 0 时，两个超链接文本都不醒目显示。这种情况不需要输入变量$lb 和$name。

当处于登录状态时（变量$isLogon 的值为 true），主导航栏的呈现效果如图 A-6 所示。



图 A-6　学生-教师子系统主导航栏（登录状态）

其中，“胡文海”是输入变量$name 的值，“同学”是输入变量$lb 的值。“进入系统”和“退出登录”是两个超链接的文本，两个超链接的 href 属性值分别设置为"enter.php"和

"logoff.php"。这种情况不需要输入变量$choice。

（2）创建 PHP 文件 student\navigation_student.php，作为能够呈现学生子系统导航栏的模块文件，用于学生子系统各页面。

该导航栏模块文件的执行要用到 2 个变量，当这些变量取不同值时会有不同的呈现效果。下面是这 2 个输入变量的变量名、含义及可能的取值：

```
$name = <姓名>          // 当前登录学生的姓名
$choice = {1|2|3}      // 用户选择状态，取 1、2 或 3
```

运行该模块文件的呈现效果如图 A-7 所示。

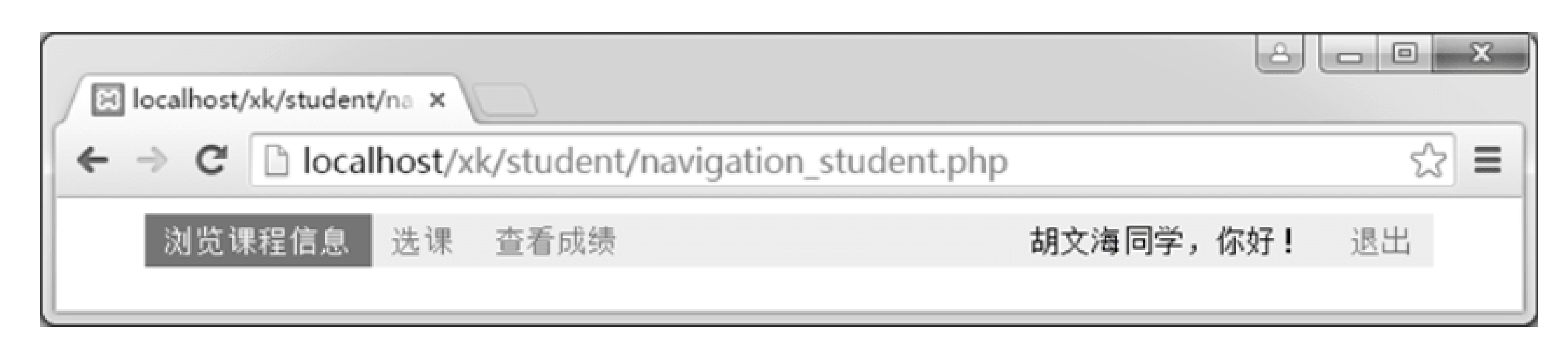


图 A-7　学生子系统导航栏

这里，“浏览课程信息”“选课”“查看成绩”和“退出”都是超链接的文本，这些超链接的 href 属性值分别设置如下：

- 浏览课程信息：course_p.php。
- 选课：elective_p.php。
- 查看成绩：score_view_p.php。
- 退出：../index.php。

这些超链接文本应该根据输入变量$choice 的取值，决定是否醒目显示。当$choice 分别取 1、2 和 3 时，超链接文本“浏览课程信息”“选课”和“查看成绩”分别醒目显示。

（3）创建 PHP 文件 teacher\navigation_teacher.php，作为能够呈现教师子系统导航栏的模块文件，用于教师子系统的各页面。

该导航栏模块文件的执行要用到 2 个变量，当这些变量取不同值时会有不同的呈现效果。下面是这 2 个输入变量的变量名、含义及可能的取值：

```
$name = <姓名>          // 当前登录教师的姓名
$choice = {1|2|3}      // 用户选择状态，取 1、2 或 3
```

运行该模块文件的呈现效果如图 A-8 所示。



图 A-8　教师子系统导航栏

这里，“课程列表”“编辑课程信息”“录入成绩”和“退出”都是超链接的文本，这些超链接的 href 属性值分别设置如下：

- 课程列表：course_teacher_p.php。
- 编辑课程：course_edit_p.php。
- 录入成绩：score_input_p.php。
- 退出：../index.php。

这些超链接文本应该根据输入变量$choice 的取值，决定是否醒目显示。当$choice 分别取 1、2 和 3 时，超链接文本“课程列表”“编辑课程信息”和“录入成绩”分别醒目显示。

A.4 实验 4：子系统主页

本实验创建学生-教师子系统的主页文件 index.php，该文件保存于教务选课系统项目 xk 中的“源文件”结点。本实验还对在实验 1 创建的页面头模块文件（header.php）进行了相应的完善。

A.4.1 目的与要求

（1）掌握 PHP 跳转语句的使用。

（2）掌握 PHP 函数定义与调用的方法，掌握 PHP 内置的日期时间函数的使用。

（3）理解页面模块化设计的思想，掌握 PHP 包含文件技术的使用。

A.4.2 实验内容

（1）利用 PHP 内置的有关日期时间函数，修改在实验 1 中创建的用于呈现页面头的模块文件 header.php，使得在呈现页面头时能动态产生并显示当前的日期和时间。

将页面头模块文件中的 logo 图标改为超链接图标，并将该超链接的 href 属性值设置为学生-教师子系统主页文件 index.php。

（2）利用 PHP 包含文件技术，创建学生-教师子系统主页文件 index.php。该文件保存于教务选课系统项目 xk 中的“源文件”结点。主页由页面头、导航栏、页面主区和页面脚组成，其呈现效果如图 A-9 所示。



图 A-9 学生-教师子系统主页

由于到本实验为止，还没有实现用户登录和会话等功能，所以还无法动态获取当前登录用户的相关信息。为此，在调用主导航栏模块文件（navigation.php）时，可以暂时将各输入变量设置为某个合适的值。对学生-教师子系统主页来说，如果处于未登录状态，输入变量$choice 应设置为 0。

A.5 实验 5：课程列表

本实验实现教师子系统导航栏中“课程列表”菜单项的相应功能，可以显示当前登录教师负责的课程列表。本实验创建的文件包括：

（1）course_teacher.php：“课程列表”模块文件，实现“课程列表”菜单项的相应功能。

（2）course_teacher_p.php：“课程列表”页面文件，页面由页面头、导航栏、页面主区和页面脚 4 部分组成，页面主区显示课程列表，由模块文件 course_teacher.php 负责呈现。

两个文件都保存于教务选课系统项目 xk 中的“源文件”结点下的 teacher 子目录。

A.5.1 目的与要求

（1）了解 PHP 访问数据库的 MySQLi 技术和 PDO 技术。

（2）掌握使用 MySQLi 扩展连接 MySQL 数据库的方法。

（3）掌握使用 MySQLi 扩展执行 SQL 语句和处理查询结果的方法技术。

（4）掌握使用 HTML <table>标签动态显示查询结果（数据表格）的方法。

（5）理解页面模块化设计的思想，掌握 PHP 包含文件技术的使用。

A.5.2 实验内容

（1）创建“课程列表”模块文件 course_teacher.php，用于显示当前登录教师负责的课程列表，每门课程显示“课程号”“课程名”和“学分”三项信息。模块文件执行的呈现效果如图 A-10 所示页面的页面主区部分。



图 A-10 教师子系统“课程列表”页面

该模块文件的执行需要当前登录教师的教师号。这里假定当前登录教师的教师号保存在输入变量$tn 中，调用该模块文件的页面文件需要事先设置该输入变量。

（2）利用 PHP 包含文件技术，创建教师子系统的“课程列表”页面文件，该文件的文件名为 course_teacher_p.php。除显示页面头、导航栏和页面脚之外，页面的主区显示当前登录教师负责的课程列表。整个页面的呈现效果如图 A-10 所示。

由于到本实验为止，还没有实现用户登录和会话等功能，所以还无法动态获取当前登录教师的相关信息。为此，在页面文件调用教师子系统导航栏模块文件、“课程列表”模块文件时，可以暂时将$name 和$tn 等输入变量设置为某位教师的姓名和教师号。

A.6　实验 6：查看成绩

本实验实现学生子系统导航栏中“查看成绩”菜单项的相应功能，可以显示当前登录学生已修课程的成绩列表，包括有关统计信息。本实验创建的文件包括：

（1）score_view.php：“查看成绩”模块文件，实现“查看成绩”菜单项的相应功能。

（2）score_view_p.php：“查看成绩”页面文件，页面由页面头、导航栏、页面主区和页面脚 4 部分组成。页面主区显示有关成绩信息，由模块文件 score_view.php 负责呈现。

两个文件都保存于教务选课系统项目 xk 中的“源文件”结点下的 student 子目录。

A.6.1　目的与要求

（1）掌握使用 MySQLi 扩展连接 MySQL 数据库的方法。

（2）掌握使用 MySQLi 扩展进行多表连接查询的方法。

（3）掌握使用 MySQLi 扩展处理缓存结果集的方法和技术。

（4）掌握使用 HTML <table>标签动态显示查询结果（数据表格）的方法。

A.6.2　实验内容

（1）创建“查看成绩”模块文件 score_view.php，用于显示当前登录学生已修课程的成绩列表，以及不及格门数和平均成绩等统计信息。这些课程的开课状态应为“结课”。每门课程显示“学期”“课程名”“学分”和“成绩”四项信息，各门课程按学期升序排序，同一学期的课程按课程号升序排序。模块文件执行的呈现效果如图 A-11 所示页面的页面主区部分。

该模块文件的执行需要当前登录学生的学号（用户名）和姓名。这里假定当前登录学生的学号保存在输入变量$sn 中、姓名保存在输入变量$name 中，调用该模块文件的页面文件需要事先设置这些输入变量。

（2）利用 PHP 包含文件技术，创建学生子系统的“查看成绩”页面文件，该文件的文件名为 score_view_p.php。除显示页面头、导航栏和页面脚之外，页面的主区显示当前登录学生已修课程的成绩列表。整个页面的呈现效果如图 A-11 所示。

由于到本实验为止，还没有实现用户登录和会话等功能，所以还无法动态获取当前登录学生的相关信息。为此，在页面文件调用学生子系统导航栏模块文件、“查看成绩”模块文件时，可以暂时将$name 和$sn 等输入变量设置为某位学生的姓名和学号。



图 A-11　学生子系统“查看成绩”页面

A.7　实验 7：注册与登录

本实验有两项任务：

（1）创建学生“注册”页面，实现学生注册功能；

（2）创建学生和教师“登录”页面，实现学生或教师登录功能。

本实验创建的文件包括：

（1）registration_p.php：注册页面文件，除显示页面头、导航栏和页面脚之外，页面主区显示注册表单，由模块文件 registration.php 负责呈现。

（2）login_p.php：登录页面文件，除显示页面头、导航栏和页面脚之外，页面主区显示登录表单，由模块文件 login.php 负责呈现。

（3）enter.php：根据登录信息有选择地进入学生子系统或教师子系统。

（4）logoff.php：退出登录状态。

这 4 个文件都保存于教务选课系统项目 xk 中的“源文件”结点。

本实验还对之前实验创建的一些文件进行了所需的修改，包括：

（1）registration.php：用于注册的模块文件，创建于实验 2。在此实现注册功能。

（2）login.php：用于登录的模块文件，创建于实验 2。在此实现登录功能。

（3）index.php：子系统主页文件，创建于实验 4。在此根据登录信息，为调用主导航栏模块文件设置输入变量。

（4）course_teacher_p.php：“课程列表”页面文件，创建于实验 5。在此根据登录信息，为调用教师子系统导航栏模块文件和“课程列表”模块文件设置输入变量。

（5）score_view_p.php："查看成绩"页面文件，创建于实验 6。在此根据登录信息，为调用学生子系统导航栏模块文件和"查看成绩"模块文件设置输入变量。

A.7.1 目的与要求

（1）掌握动态表单（包括错误信息的呈现，以及输入数据的回显）的制作技术。
（2）掌握获取请求参数值的方法。
（3）掌握使用 MySQLi 扩展访问和更新数据库的方法和技术。
（4）理解 Web 应用中会话的概念。
（5）掌握启动或恢复会话的方法，以及设置会话变量和访问会话变量的技术。
（6）掌握清除会话变量的技术。
（7）理解 PRG 模式，掌握页面重定向技术。

A.7.2 实验内容

（1）修改实验 2 中创建的、用于注册的模块文件 registration.php，具体要求如下：

① 对用户提交的注册信息进行验证。

- 必填项（带*者）不能为空；
- 两次输入的密码需一致；
- 出生日期和 Email 地址应该满足相应的格式；
- 用户名（学号）在 student 表中并不存在相应的行。

② 如果注册信息未通过验证，就重新显示注册表单。此时应回显用户原先输入的值，并显示相应的错误信息，以便用户修改。

③ 如果注册信息通过了验证，则根据注册信息在 student 表中插入相应的行，然后重定向至子系统主页 index.php。

（2）利用 PHP 包含文件技术，创建系统的注册页面文件 registration_p.php。页面由页面头、导航栏、注册表单和页面脚组成，如图 A-12 所示。

（3）修改实验 2 中创建的、用于登录的模块文件 login.php，具体要求如下。

① 对用户提交的登录信息进行验证。

- 用户名（学号或教师号）和密码不能为空；
- 在 student 表或 teacher 表中存在相应的行，其学号或教师号与用户提交的用户名一致，其密码与用户提交的密码一致。

② 如果登录信息未通过验证，就重新显示登录表单。此时应回显用户原先输入的值，并显示相应的错误信息，以便用户修改。

③ 如果登录信息通过了验证，那么就注册 3 个会话变量：user 为用户提交的用户名（学号或教师号），lb 为用户提交的身份类别（字符串"同学"或"老师"），name 为学生或教师的姓名。然后重定向至子系统主页 index.php。

（4）利用 PHP 包含文件技术，创建系统的登录页面文件 login_p.php。页面由页面头、导航栏、登录表单和页面脚组成，如图 A-13 所示。

（5）修改实验 4 中创建的子系统主页文件（index.php）。根据已存在的会话变量的值（登录信息）设置调用主导航栏模块文件（navigation.php）所需的输入变量：

学生注册

localhost/xk/registration_p.php

欢迎使用 教学选课系统！

2016年12月13日
星期二
下午 09:33:58

登录 注册

请注册

用户名*

密码*

确认密码*

姓名*

性别 男 女

出生日期

Email*

立即注册

Copyright©2016 首都经济贸易大学 信息学院 电话:1234567 邮箱:abcde@cued.edu.cn

图 A-12 学生注册页面

localhost/xk/login_p.php

欢迎使用 教学选课系统！

2016年12月13日
星期二
下午 09:35:36

登录 注册

请登录

用户名

密 码

学生 教师

确 认

Copyright©2016 首都经济贸易大学 信息学院 电话:1234567 邮箱:abcde@cued.edu.cn

图 A-13 学生-教师登录页面

① 如果存在会话变量 user，则将输入变量$isLogon 的值设置为 true，同时将输入变量$name 的值设置为同名的会话变量的值，将输入变量$lb 的值设置为同名的会话变量的值。

② 如果不存在会话变量 user，则将输入变量$isLogon 的值设置为 false，将输入变量$choice 的值设置为 0。

（6）修改实验 5 中创建的“课程列表”页面文件（course_teacher_p.php）。根据已存在的会话变量的值（登录信息）设置相关变量，具体要求如下：

① 如果不存在名为 lb 的会话变量，或者该会话变量的值不为“老师”，则跳转至子系统主页面 index.php。

② 在调用教师子系统导航栏模块文件（navigation_teacher.php）之前，将输入变量$name 的值设置为同名的会话变量的值，输入变量$choice 的值仍设置为 1。

③ 在调用“课程列表”模块文件（course_teacher.php）之前，将输入变量$tn 的值设置为名为 user 的会话变量的值。

（7）修改实验 6 中创建的“查看成绩”页面文件 score_view_p.php。根据已存在的会话变量的值（登录信息）设置相关变量，具体要求如下：

① 如果不存在名为 lb 的会话变量，或者该会话变量的值不为“同学”，则跳转至子系统主页面 index.php。

② 在调用学生子系统导航栏模块文件（navigation_student.php）之前，将输入变量$name 的值设置为同名的会话变量的值，输入变量$choice 的值仍设置为 3。

③ 在调用“查看成绩”模块文件（score_view.php）之前，将输入变量$sn 的值设置为名为 user 的会话变量的值。

（8）实现“退出登录”和“进入系统”功能，具体要求如下：

① 创建 PHP 文件 logoff.php，其功能是终止当前会话（销毁会话变量），然后重定向至子系统主页 index.php。

② 创建 PHP 文件 enter.php，其功能是判断当前登录用户的身份类别，若是“老师”就跳转至教师子系统的“课程列表”页面（course_teacher_p.php）；否则跳转至学生子系统的“查看成绩”页面（score_view_p.php）。

A.8　实验 8：编辑课程信息

本实验实现教师子系统导航栏中“编辑课程信息”菜单项的相应功能，可以对当前登录教师所负责的课程进行编辑，包括输入和编辑课程描述、上传课程大纲。

本实验创建的文件有以下两种。

（1）course_edit.php：“编辑课程信息”模块文件，实现菜单项的相应功能。

（2）course_edit_p.php：“编辑课程信息”页面文件，显示页面头、导航栏、页面主区和页面脚，其中页面主区由模块文件 course_edit.php 负责呈现。

两个文件都保存于教务选课系统项目 xk 中的“源文件”结点下的 teacher 子目录。

A.8.1　目的与要求

（1）掌握区分请求源（同一个页面的不同元素）的方法。

（2）掌握使用 MySQLi 扩展访问和更新数据库的方法和技术。

（3）掌握会话启动或恢复的方法，以及访问会话变量的技术。

（4）掌握文件上传表单的设计方法。

（5）掌握 PHP 中访问及处理上传文件的方法和技术。

（6）掌握利用数组数据、动态产生选择列表选项的方法。

A.8.2　实验内容

（1）创建“编辑课程信息”模块文件 course_edit.php，用于编辑、上传一门课程的课程描述和课程大纲，每位教师只能编辑自己负责的课程信息。

该模块文件的执行需要当前登录教师的教师号。这里假定当前登录教师的教师号保存在输入变量$tn 中，调用该模块文件的页面文件需要事先设置该输入变量。

根据用户请求方式不同，该模块文件会进行不同的数据处理和呈现不同的内容。当用户以 GET 方法（不带请求参数）请求该模块文件（或其所在的页面文件）时，该模块文件将获取当前教师负责的课程，并呈现如图 A-14 所示的表单。

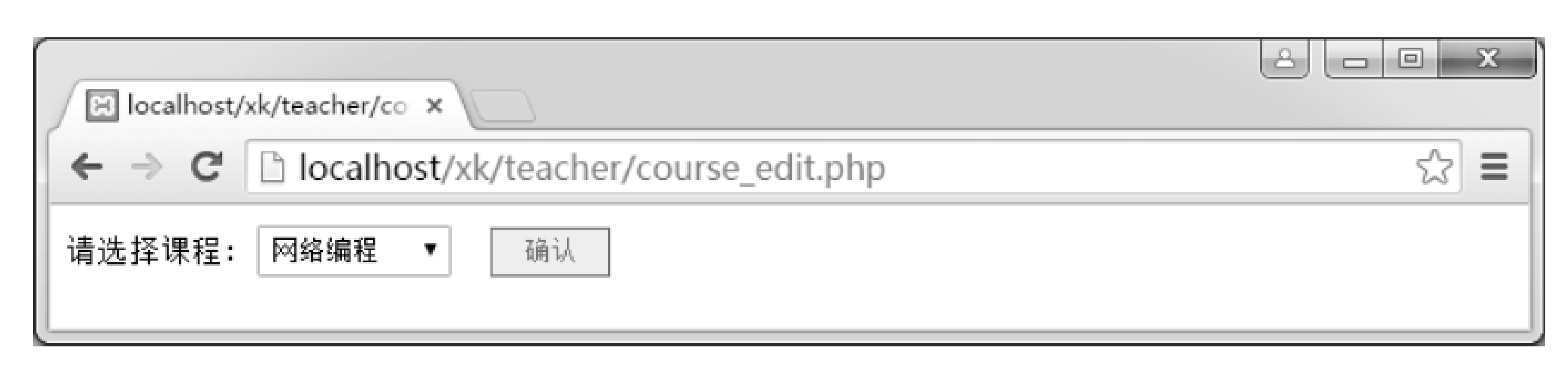


图 A-14　编辑课程信息模块——选择课程

其中，选择列表选项应该包含当前教师负责的课程名。当用户选择一门课程并单击“确认”按钮后，将向服务器发送一个 POST 请求（带一个包含课程号的请求参数），请求该模块文件（或其所在的页面文件）。此时，该模块文件将获取选定课程的信息，并呈现如图 A-15 所示的表单。



图 A-15　编辑课程信息模块——编辑表单

这里，用户可以输入或编辑课程描述、上传课程大纲。当单击“提交”按钮后，将向服务器发送一个 POST 请求（包含该课程的课程描述和课程大纲），请求该模块文件（或其所在的页面文件）。此时，该模块文件会把用户提交的课程相关信息进行保存处理，然后呈现处理的结果信息，如图 A-16 所示页面的主区部分。

这里，如果用户单击“继续编辑其他课程”，将向服务器发送一个 GET 请求（不带参数），请求该模块文件（或其所在的页面文件）。用户可以继续选择并编辑课程信息。

（2）利用 PHP 包含文件技术，创建“编辑课程信息”页面文件 course_edit_p.php。页面由页面头、导航栏、页面主区和页面脚组成，其中页面主区由“编辑课程信息”模块文件 course_edit.php 负责呈现。呈现效果如图 A-16 所示。



图 A-16 “编辑课程信息”页面——显示处理结果信息

在调用教师子系统导航栏模块文件（navigation_teacher.php）和“编辑课程信息”模块文件（course_edit.php）之前，应该根据当前登录信息为它们设置相应的输入变量。

A.9 实验 9：浏览课程信息

本实验实现学生子系统导航栏中“浏览课程信息”菜单项的功能，可以分页显示课程基本信息、显示课程详细信息、下载课程大纲。本实验创建的文件有以下四种。

（1）course_list.php：“课程基本信息”模块文件，可以分页显示所有课程的基本信息。

（2）course_detail.php：“课程详细信息”模块文件，可以显示某门课程的详细信息。

（3）down_outline.php：可以下载指定课程的大纲。

（4）course_p.php：“浏览课程信息”页面文件，显示页面头、导航栏、页面主区和页面脚，其中页面主区会根据请求上下文决定：是包含模块文件 course_list.php 以呈现课程基本信息列表，还是包含模块文件 course_detail.php 以呈现课程详细信息。

这些文件都保存于教务选课系统项目 xk 中的“源文件”结点下的 student 子目录。

A.9.1　目的与要求

（1）掌握在 URL 中动态添加请求参数的方法。
（2）掌握分页显示的技术。
（3）掌握文件下载的方法和技术。

A.9.2　实验内容

（1）创建“课程基本信息”模块文件 course_list.php，可以分页显示 course 表中所有课程的基本信息，每门课程包括课程号、课程名、学分和负责教师 4 项信息，如图 A-17 所示页面的主区部分。其中，最左侧的各课程号是超链接文本。单击某个课程号将产生一个 GET 请求，请求“浏览课程信息”页面文件 course_p.php，并以当前页码和当前课程号作为请求参数，参数名称分别为 p 和 cn。



图 A-17　“浏览课程信息”页面——课程基本信息列表

该模块文件（或所在的页面文件）应该通过 GET 方法请求，并可携带一个名称为 p 的请求参数，以指示该模块文件应该显示哪一页的课程基本信息列表。如果不包含名称为 p 的请求参数，模块文件应该显示第 1 页的课程基本信息列表。

（2）创建“课程详细信息”模块文件 course_detail.php，可以显示指定课程的详细信息，如图 A-18 所示页面的页面主区部分。在课程详细信息中，最下面课程大纲一栏显示的是课程大纲文件的文件名，是一个超链接文本。单击该文件名将产生一个 GET 请求，请求 PHP 文件 down_outline.php，并把课程大纲文件名作为请求参数。在课程详细信息下方，有一个超链接文本“返回课程列表”，单击该超链接可以返回之前的课程基本信息列表。

该模块文件（或所在的页面文件）应该通过 GET 方法请求，并携带一个名称为 p 的请求参数和一个名称为 cn 的请求参数。请求参数 cn 指示该模块文件应该显示哪一门课程的详细信息。请求参数 p 指示当返回时应该显示哪一页的课程基本信息列表。



图 A-18 “浏览课程信息”页面——课程详细信息

（3）创建 PHP 文件 down_outline.php。该 PHP 文件应该通过 GET 方法请求，并携带一个包含课程大纲文件名的请求参数。该 PHP 文件首先获取该请求参数，然后下载指定课程的课程大纲文件。

（4）利用 PHP 包含文件技术，创建“浏览课程信息”页面文件 course_p.php。页面由页面头、导航栏、页面主区和页面脚组成。在请求该页面文件时，如果没有包含名称为 cn 的请求参数，那么页面主区由模块文件 course_list.php 负责呈现；否则页面主区由模块文件 course_detail.php 负责呈现。其呈现效果如图 A-17 和 A-18 所示。

在调用学生子系统导航栏模块文件（navigation_student.php）之前，应该根据当前登录信息为它设置相应的输入变量。

A.10 实验 10：录入成绩

本实验实现教师子系统导航栏中“录入成绩”菜单项的功能，可以显示、输入和提交当前登录教师所讲授课程的成绩。本实验创建的文件有以下两种。

（1）score_input.php：“录入成绩”模块文件，实现菜单项所要求的相关功能。

（2）score_input_p.php：“录入成绩”页面文件，显示页面头、导航栏、页面主区和页面脚，其中页面主区由模块文件 score_input.php 负责呈现。

两个文件都保存于教务选课系统项目 xk 中的“源文件”结点下的 teacher 子目录。

A.10.1 目的与要求

（1）掌握使用 MySQLi 扩展访问和更新数据库的方法和技术。
（2）理解 PHP 数组的概念，区分数字索引数组和关联数组。
（3）掌握数组创建、数组元素操作和数组遍历的方法和技术。
（4）掌握利用数组数据、动态产生表单元素的方法，实现批量数据录入。

A.10.2 实验内容

（1）创建“录入成绩”模块文件 score_input.php，用于输入、编辑或查看学生选修完某门课程后的成绩。每位教师只能输入和编辑自己讲授的处于“教学”状态（status 字段的值为'2'）的课程的成绩，或者查看自己讲授的处于“结课”状态（status 字段的值为'3'）的课程的成绩。

该模块文件的执行需要当前登录教师的教师号。这里假定当前登录教师的教师号保存在输入变量$tn 中，调用该模块文件的页面文件需要事先设置该输入变量。

根据用户请求方式不同，该模块文件会进行不同的数据处理和呈现不同的内容。当用户以 GET 方法（不带请求参数）请求该模块文件（或其所在的页面文件）时，该模块文件将获取当前教师讲授的处于“教学”或“结课”状态的开课课程，并呈现如图 A-19 所示的表单。

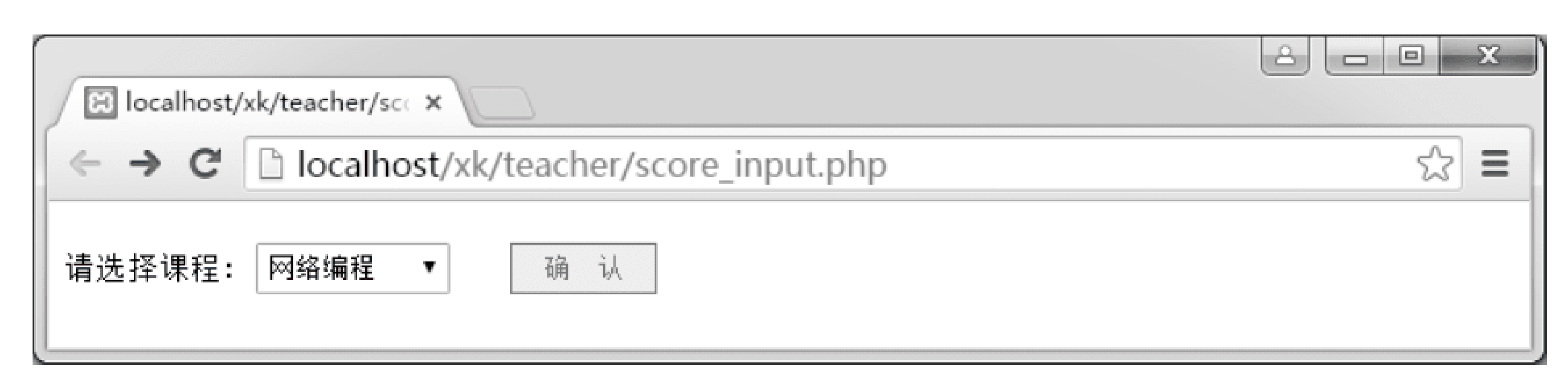


图 A-19 录入成绩模块——选择课程

其中，选择列表选项应该包含当前教师讲授的开课（状态为“教学”或“结课”）课程名。当用户选择一门课程，单击“确认”按钮后，将向服务器发送一个 POST 请求（带一个包含开课号的请求参数，参数名为 lid），请求该模块文件（或其所在的页面文件）。处理请求时，该模块文件首先会将开课号注册为一个会话变量，然后获取选修该开课课程的所有学生及成绩信息。

如果该开课课程处于“教学”状态，该模块文件将呈现如图 A-20 所示的表单，教师可以输入或编辑选修该课程的学生的成绩。

如果该开课课程处于“结课”状态，该模块文件将显示选修该课程的所有学生的成绩信息，其效果如图 A-21 所示页面的主区部分。此时，教师不能输入或编辑相关的成绩。

在输入和编辑学生成绩表单中，当单击“保存成绩”按钮后，将向服务器发送一个 POST 请求（包含所有成绩），请求该模块文件（或其所在的页面文件）。此时，该模块文件会把用户提交的成绩进行保存处理，然后再次呈现该表单。如果单击“提交成绩”按钮，将向服务器发送一个 POST 请求（包含所有成绩），请求该模块文件（或其所在的页面文件）。此时，该模块文件除了会把用户提交的成绩进行保存处理，还会改变该开课课程的状态，



图 A-20　录入成绩模块——输入指定课程的成绩



图 A-21　“录入成绩”页面——显示指定课程成绩

由“教学”状态变为“结课”状态，然后呈现选修该课程的所有学生的成绩信息（成绩已不能再修改）。

（2）利用 PHP 包含文件技术，创建“录入成绩”页面文件 score_input_p.php。页面由页面头、导航栏、页面主区和页面脚组成，如图 A-21 所示。其中页面主区由“录入成绩”模块文件（score_input.php）负责呈现。

页面文件在调用教师子系统导航栏模块文件（navigation_teacher.php）和“录入成绩”模块文件（score_input.php）之前，应该根据当前登录信息为它们设置相应的输入变量。

A.11 实验 11：选课

本实验实现学生子系统导航栏中“选课”菜单项的功能，可以分页显示可选的开课课程列表，允许当前登录学生选择自己要选修的课程。本实验创建的文件有以下两种。

（1）elective.php：“选课”模块文件，实现菜单项所要求的相关功能。

（2）elective_p.php：“选课”页面文件，显示页面头、导航栏、页面主区和页面脚，其中页面主区由模块文件 elective.php 负责呈现。

两个文件都保存于教务选课系统项目 xk 中的“源文件”结点下的 student 子目录。

A.11.1 目的与要求

（1）掌握使用 MySQLi 扩展访问和更新数据库的方法和技术。

（2）掌握数组创建、数组元素操作和数组遍历的方法和技术。

（3）掌握数组联合、拆分等操作的方法和技术。

（4）掌握利用数组数据、动态产生表单元素的方法，实现批量数据录入。

（5）掌握产生混合 GET 和 POST 请求的方法和技术。

A.11.2 实验内容

（1）创建“选课”模块文件 elective.php，可以分页显示所有处于“选课”状态的开课课程列表，学生可以点击课程右侧的复选框选择自己要选修的课程。其呈现效果如图 A-22 所示页面的主区部分。



图 A-22 “选课”页面—可选开课课程列表

该模块文件的执行需要当前登录学生的学生号。这里假定当前登录学生的学生号保存在变量$sn 中，调用该模块文件的页面文件需要事先设置该输入变量。

下面是实现该模块文件的一些具体要求。

① 当用户以 GET 方法（不带请求参数）请求该模块文件（或其所在的页面文件）时，模块文件应将当前页码设置为 1，即显示可选开课课程列表的第 1 页。

允许学生多次访问、选择和更改。每次访问时，可选开课课程列表右侧的复选框应该能够反映该学生之前访问所做的选择。

② 翻页导航栏中的页码是一些提交按钮。当单击翻页导航栏中的页码按钮时，将产生对该模块文件（或其所在的页面文件）的 POST 请求，其中所选页码应该作为 GET 请求参数，当前页的选择状态数据应该作为 POST 请求参数。此时模块文件首先需要对提交的状态数据进行缓存，以便当再次翻回到该页时，能够呈现出最新的状态。然后，模块文件应该呈现可选开课课程列表的指定页。

③ 当单击“确认提交”按钮时，将产生对该模块文件（或其所在的页面文件）的 POST 请求，并提交当前页的选择状态数据。此时模块文件首先需对提交的状态数据进行缓存，然后再将本次访问所做的所有选择和更改反映到 elective 表中，即所选择的任何一门课在该表中都有相应的一条记录：学号（sn）为当前登录学生的学号、开课号（lid）为所选开课课程的开课号、成绩（score）为 NULL。当上述处理完成后，模块文件显示相应的提示信息，如图 A-23 所示页面的主区部分。

④ 当用户单击提示信息中的“选课表”超链接时，将产生对该模块文件（或其所在的页面文件）的 GET 请求（不带参数），此时模块文件可将当前页码设置为 1，即显示可选开课课程列表的第 1 页。

（2）利用 PHP 包含文件技术，创建“选课”页面文件 elective_p.php。页面由页面头、导航栏、页面主区和页面脚组成，如图 A-22 和图 A-23 所示。其中页面主区由“选课”模块文件 elective.php 负责呈现。



图 A-23 “选课”页面——确认提交提示信息

在页面文件调用学生子系统导航栏模块文件（navigation_student.php）和“选课”模块文件（elective.php）之前，应该根据当前登录信息为它们设置相应的输入变量。

参 考 文 献

[1] WELLING L, THOMSON L. PHP 和 MySQL Web 开发[M]. 4 版. 武欣，译. 北京：机械工业出版社，2009.

[2] 张亚东，高红霞. PHP+MySQL 全能权威指南[M]. 北京：清华大学出版社，2012.

[3] YORK R. 精通 jQuery Web 开发[M]. 2 版. 李周芳，译. 北京：清华大学出版社，2015.

[4] 陶国荣. jQuery 权威指南[M]. 北京：机械工业出版社，2013.

[5] ULLMAN L. 深入理解 PHP[M]. 3 版. 季国飞，等译. 北京：机械工业出版社，2014.

[6] GROSS C. Ajax 模式与最佳实践[M]. 李锟，等译. 北京：电子工业出版社，2007.